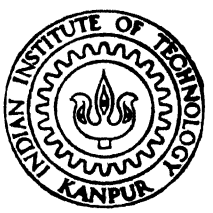
Spatio - temporal Pattern Recognition with Artificial Neural Networks

by

YADAVALLI SITARAM



DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

APRIL, 1991

Spatio-temporal Pattern Recognition with Artificial Neural Networks

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of
Master of Technology

by
Yadavalli Sitaram

to the
DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

April 1991

CERTIFICATE

It is certified that the work contained in the thesis entitled *Spatio-temporal Pattern Recognition with Artificial Neural Networks* by *Yadavalli Sitaram* has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

P.R.K. Rao
Professor

S.K. Mullick
Professor

April 1991

Department of Electrical Engineering
Indian Institute of Technology, Kanpur

ABSTRACT

In this thesis we discuss various issues in spatio-temporal pattern recognition using connectionist paradigms. We present a new back-propagation algorithm for training a class of neural networks for recognising patterns in space and time. The network we propose is a linear filter artificial neuron system. The basic unit uses an artificial neuron followed by a linear time-invariant or time-varying causal filter. Filter could be of recursive or non-recursive type. The neural unit uses a quadratic time-weighted output deficit error function which has to be minimised in the training phase. We present a learning algorithm to minimise the error by suitably adapting the filter parameters as well as the weights of the network using a gradient based error minimisation technique. The constraints on the filter parameters to ensure stability have also been derived. Computer simulations have been carried out extensively to evaluate the performance of the system on temporal and spatio-temporal pattern recognition in a variety of cases. Results on robustness of the network as well as convergence of the algorithm based on empirical observations have also been discussed.

ACKNOWLEDGEMENTS

I wish to express my deep and sincere thanks to Profs. P.R.K.Rao and S.K.Mullick for their encouragement, guidance and unceasing enthusiasm throughout this thesis work and beyond.

I also thank Prof. K.R.Srivatsan and my friend Satish Mantripragada for allowing me to use the hp workstation *Kalyan* housed in our lab with full freedom. My thanks are also due to my friend P.V.K.Reddy for allowing me to use the facilities in the HVDC lab and for the numerous discussions we had on neural networks.

Finally, my word of thanks to all those who made this work possible and the experience so rewarding and enjoyable.

CONTENTS

Chapter 1
Introduction

1. Introduction

The twentieth century has seen an unprecedented growth in information processing in every conceivable field of human activity. Such a growth in information processing has been feasible since the advent of electronic intruments and calculating machines in the early years of this century. With the advent of compact high speed electronic computers using solid-state technology in the sixties, more complex varieties of processing capabilities emerged. Now with drastic minimisation of physical dimensions of electronic ciruits using integrated circuit technology it has become possible to design circuits with more than a million "electronic switches" or transistors. This has made it practical to look at parallel distributed processing (PDP) techniques which were earlier considered to be unconventional and impractical because of the huge number of processing elements they need. We shall discuss PDP a little later.

Pattern recognition, which was hitherto a forte of biological systems alone, soon came to be examined as one of the several information processing tasks a computer may handle. Pattern recognition entails a discussion on our conceptualization of what may constitute intelligence. Researchers in the fifties and sixties examined the whole issue of machine recognition of objects, and concurrently emerged a plethora of concepts and questions in a field that has since come to be known as artificial intelligence or AI. Most of the artificial intelligence work progressed on the Von Neumann stored-program-stored-data sequential computing model, in which, a program sequentially processes the data stored in its memory. Thus, artificial intelligence encouraged the algorithmic approach of manipulating object data for the purpose of pattern recognition.

Researchers using the AI approach to pattern recognition have based their algorithms on theories of cognition and theories of mental representation of objects and concepts. These theories depend on specific models that have been built for explaining higher mental capacities of human beings, a couple of illustrative examples being spatial pattern recognition, speech recognition and the ability to communicate by means of a language. This is the top-down approach which calls for an explicit model building exercise - a model to describe what constitutes a particular mental process.

Let us take for instance the task of recognising spatio-temporal patterns which is the topic of main interest in this thesis. Spatio-temporal patterns are patterns or events that occur in time and space. A specific example of such pattern recognition activity is that of recognizing speech. Researchers have developed complex models for speech and language. One such model is the hidden Markov model which treats speech as a doubly stochastic process. Researchers have used computers to perform speech recognition based on this model along with a model for the language in which a particular piece of speech is uttered. Using statistical and syntactic techniques incorporated in their algorithms, they have been successful in making computers perform speech recognition.

In early fifties some researchers took an entirely different approach to perform pattern recognition and emulate human intelligence, an approach that has come to be known by a variety of names. It is the neural network or connectionist approach to pattern recognition. In neural network paradigms the basic architecture comprises of artificial neurons interconnected according to a predecided scheme. These artificial neurons are the mathematical descriptors of the biological neuron in some sense. An artificial neuron is, essentially a linear combiner followed by a non-linearity. These units are interconnected amongst

themselves. Such interconnections are the mathematical analogs of the biological synapses between neurons, and the synaptic strengths are modelled using the weights of the linear combiners. The system is trained in one of the several available ways. During the training period the synaptic weights are modified suitably in accordance with a learning rule that is used with the aim of maximising the pattern-recognizing capacity of the system under consideration. The neurons in the network work in parallel, changing their state depending on their inputs. Such a parallel operation of neurons in an artificial neural network has put these paradigms into what have come to be known as **Parallel Distributed Processing** schemes or PDP, in short.

In connectionist systems there is no explicit definition of what constitutes intelligence. Instead, it is entirely implicit in the synaptic efficacies or weights of the neural network and the particular architecture that is employed. By themselves, these weights individually have not yet yielded any explanation to researchers for the seemingly intelligent behaviour that neural networks demonstrate. It is only collectively that these neurons carry out the tasks that the system is intended to. This is then, the bottom-up approach, in constrast to the top-down approach emphasised by AI - a contrast between the implicit and explicit modes of description.

However, connectionist notions received a severe blow in the sixties when Minsky and Papert [32] proved the inadequacy of the single layered Perceptron of Rosenblatt [47] to perform even the simplest of recognition tasks. The Perceptron has been proved to be incapable of synthesizing even the XOR function. But, in 1982 Hopfield demonstrated some simple applications of recurrent networks which rejuvenated the interest in neural networks. In fact, the pioneering breakthroughs of Hopfield [22], Kohnen [26], Grossberg [16], [17], Fukushima [13], [14], [15] and

Rummelhart and McClelland [48] in the early eighties, collectively resucitated the connectionist notions of pattern recognition. It was also seen that the traditional AI approach to pattern recognition had not yielded much success even after twenty odd years of research and experimentation. Further, the connectionist systems of Kohonen, Hopfield and others were more complex than the Perceptron and performed extremely well on pattern recognition tasks. That led to this renewed interest in neural networks.

Objects of a pattern recognition task can be dichotomized into two categories namely, static patterns or objects and temporal patterns or events. Events are actions associated with objects, for example, the movement of a robot arm or spoken words in a speech recognition scheme. Much work has been done on both categories of patterns. However, all of traditional statistical or syntactic pattern recognition schemes are not only specific to the said categories of objects but also to the specific tasks and to the signals associated with the object data. While character recognition paradigms use syntax based segmentation and processing, speech recognition involves application of techniques like hidden Markov modeling - an entirely different paradigm.

Then, the question arises as to how the brain is able to perform all such recognition tasks associated with various objects and events in a single framework of connectionist paradigms. Precisely this, has been the motivation for using connectionist models for pattern recognition. Besides, cognitive tasks like speech understanding, vision, motor control of muscular apparatus, etc. are not precisely defined due to their close interaction with a highly uncertain and variable environment, and it has proved difficult to achieve even moderately good results by standard syntactic or statistical techniques. Main drawback is, of course, the fact that diverse schemes have to be used in the traditional approach.

As mentioned earlier, in traditional methods of syntactic or statistical recognition based on AI, a model for the input is assumed, to begin with. For instance the picture of a human face is considered as an image with certain features or components like the nose, the eyes, the mouth and so on while in a character recognition task the images are dissected or transformed in order to recognise strokes, vertical and horizontal bars, closed and open curves and other geometric shapes. On the other hand, in connectionist paradigms such modeling, though useful is not always necessary. Such preliminary feature extraction is called preprocessing. Almost always people have sought to eliminate this phase because neural networks are inherently very powerful and the same system can often be used to recognise a human face as well as a handwritten character upon suitable training.

While the preprocessing stage is kept minimal in static pattern recognition, most researchers have had to employ it in case of temporal pattern recognition. This is one thing we tried eliminating in the model we have presented in this thesis.

Furthermore, if connectionist schemes of pattern recognition are inherently parallel, the traditional syntactic and statistical schemes are more or less sequential in nature and the number of dependencies can be high enough to impede processing considerably which results in performance degradation in many recognition tasks where quick responses are sought.

II. Organisation of the thesis

The thesis is divided into five chapters as described below. In chapter 2 we look at various approaches to spatio-temporal pattern recognition that many researchers have put forth. We also discuss some ideas regarding the use self-

organising networks for spatio-temporal pattern analysis, especially speech segmentation. Finally, we discuss some ideas which led to the design of a network for spatio-temporal patterns, which is the main part of this thesis.

In chapter 3 we propose a network for spatio-temporal pattern recognition called ALFANS, standing for A Linear Filter Artificial Neuron System. We derive the learning algorithm for this network and discuss the constraints on various parameters in ALFANS.

In chapter 4 we present the results of our simulations of ALFAN system. We present some examples of the tasks that the network can handle. We draw our conclusions based on the results we obtained and make some critical observations.

In chapter 5 we discuss the scope for further research in this area and comment on some application domains for which ALFANS is a potential candidate.

Chapter 2
Spatio-Temporal Pattern Recognition

I. Introduction

Neural networks [29] have been successfully used for a variety of tasks involving spatial patterns. These include tasks such as character recognition and regeneration, recognition of human faces, segmentation of images, image thinning amongst others [14], [17], [23].

Researchers have considered neural network structures used for static spatial patterns to also recognise temporal patterns such as speech. However, even when the size of such networks is allowed to grow, it is often difficult to train static pattern recognition machines to adapt to recognise temporal variations [52].

In temporal pattern recognition, in contrast to, static spatial pattern recognition, the entire time pattern is not available simultaneously. To overcome this limitation, different time samples could be appropriately delayed and stored, and the recognition task carried out with these stored static patterns. One instance of this method is the music-box approach [11]. Feed-forward networks have often been used for speech recognition by converting temporal variations into static spatial patterns [6]. However, the delay associated with the conversion and storage task may not be acceptable in some applications.

Before we proceed to review the current state of research in neural networks or the so called connectionist architectures for spatio-temporal pattern recognition let us look at one of the traditional appraoaches that has been in use for speech recognition for several years now. A consideration of that method

which is often described as the hidden Markov modeling technique, enables us to understand some of the key notions associated with temporal pattern recognition, especially of speech, and to appreciate the differences between the traditional schemes and the connectionist schemes.

II. Hidden Markov Modeling of speech

The first step in the conventional methods for temporal pattern recognition like speech involves explicit characterization of the input signal in terms of a signal model. Signal models are of great help because they not only provide a theoretical description of the signal but also work fairly well in practice and are useful in realizing prediction systems, recognition systems, identification systems [42] etc. In contrast, neural network paradigms do not call for explicit signal modeling. Any such modeling incorporated through preprocessing of input signals may lead to improved performance or reduce the neural network complexity.

The signal models may be broadly divided into two categories, namely deterministic and stochastic models. In hidden Markov modeling (HMM) a doubly stochastic model [42] is assigned to the signal. It is a Markov model in which an observation sequence is governed by a probabilistic mapping of a state sequence which itself is a Markovian process that is not directly observable.

An HMM is characterized by the following:

1) A state space

2) An observation space.

3) The state transition probability distribution A.

4) The conditional probability distribution of observations given the state B

5) The initial state distribution π.

The three fundamental problems of HMM [42] which form the basis of speech

recognition are the following:

1) For each observation sequence $O=O_1O_2 \ldots O_T$, and a given model $\lambda=(A,B,\pi)$, to efficiently compute $P(O|\lambda)$, the probability distribution of the observation sequence.

2) Given an observation sequence O, and the model λ, to choose a corresponding state sequence Q which is optimal in some meaningful sense.

3) To determine the model parameters $\lambda=(A,B,\pi)$ that maximize $P(O|\lambda)$ for a given observation sequence.

Problem 1 is the analysis problem in which, given a model and an observation sequence, we estimate the probability distribution which led to that observation sequence. In problem 2 we attempt to uncover the hidden part of the model to learn about the structure of the model and find the optimal state sequences for continuous speech recognition etc. Problem 3 is the **"training problem"** in which we attempt to optimize the model parameters so as to best describe how a given observation sequence was generated. An observation sequence which is used to adjust the model parameters is called a training sequence.

Consider a simple isolated word recognition system based on HMMs. For each word of a W word vocabulary we want to design a separate N-state HMM. We represent the speech signal of a given word as a time sequence of a coded spectral vectors. Assume that the coding is done using a codebook of M unique spectral vectors making it an M alphabet system which means that each observation is the index of the spectral vector closest to the original speech in some sense. Hence, for each word in the vocabulary, there exists a training sequence consisting of a number of repetitions of sequences of codebook indices of the word. Firstly, we make the word models by using the solution to problem 3 to optimally estimate the model parameters for each model from real life speech data.

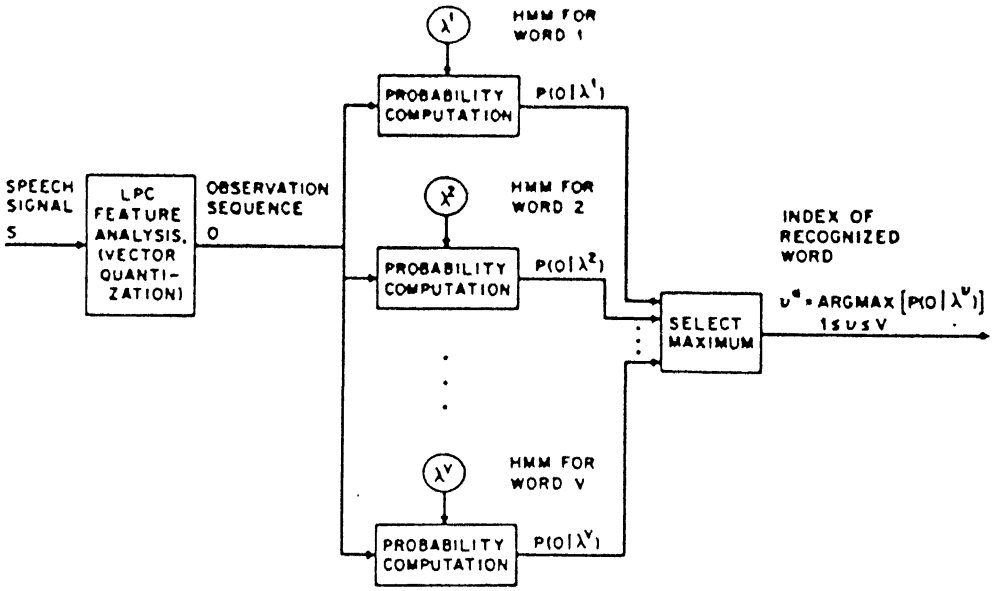


Figure 2.1 (a) Block diagram of an isolated word HMM recognizer. (Reproduced from [42])

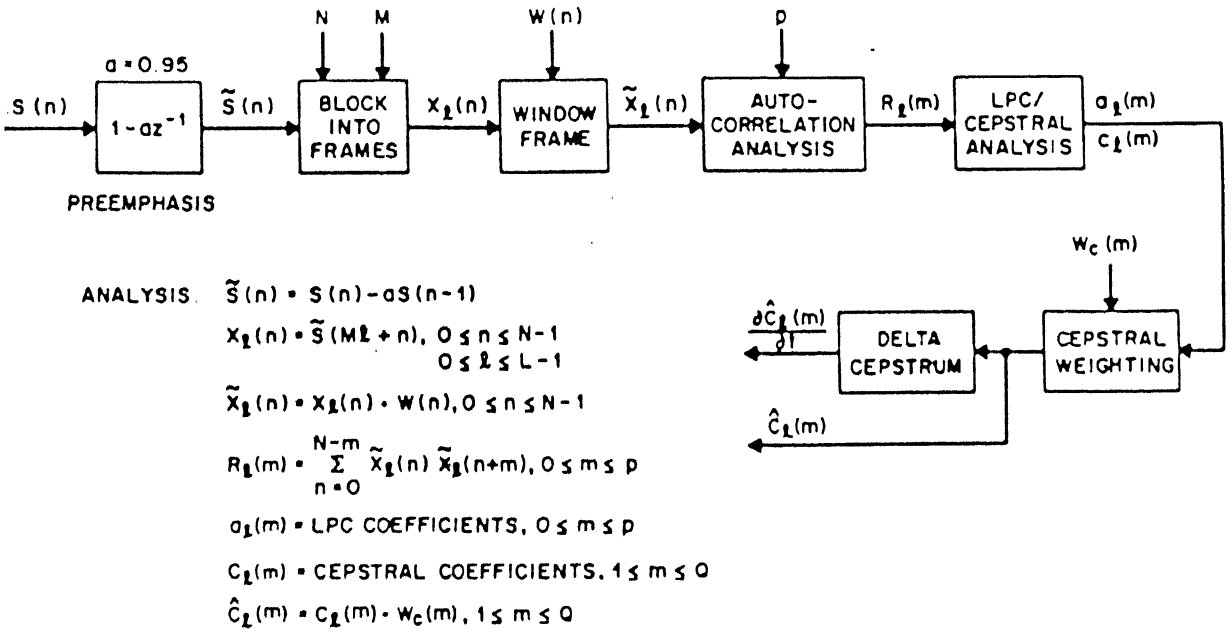


Figure 2.1 (b) Block diagram of the computations required in the front end feature analysis of the HMM recognizer. (Reproduced from [42])

This is the learning phase. We use the solution to problem 2 to segment each of the word training sequences into states in order to study the properties of spectral vectors that lead to the observations occuring in each state. This allows us to improve the model by increasing the number of states or change the codebook size, etc., to improve its capability of modeling spoken words. Finally, using the solution to problem 2 we perform recognition of an unknown word to establish as to which among the W word models is the most likely candidate from which this unknown word could have most probably been generated. The fig. 2.1(a) shows the block diagram of the computations required in the front end feature analysis of an HMM recogniser, while the block diagram of an isolated word HMM recognizer is shown in fig. 2.1(b).

III. Overview of literature in connectionist architectures for temporal pattern classification

A. NETtalk

With the great successes achieved in the recognition of static patterns using artificial neural networks, researchers started experimenting with connectionist architectures for spatio-temporal pattern recognition in 1986-87. NETtalk by Sejnowski and Rosenberg is an early effort in that direction. NETtalk uses a simple back-propagation network that reads a piece of English text character by character and reads it aloud, pronouncing it with good accuracy. They trained the network to produce speech sounds corresponding to a printed text whose characters were presented to the network one at a time, through a window with a fixed field of view. The field of view spanned the central letter alongwith 2 or 3 letters on either side of it. These extra letters were for providing contextual information to the network. The speech sound produced was

based on the phonetic descriptor that was generated at any instant corresponding to the letter in the centre of the view window aided by other adjacent letters in the field of view.

NETtalk used an heirarchical back-propagation system with three layers. Corresponding to each letter in the field of view there was an input network. All the input networks were in parallel followed by a hidden layer and an outer layer which generated the phoneme descripters. The training system was standard static back-propagation learning [48]. The network at first learnt to babble like a child and finally, became proficient in speaking with correct pronounciations and could pronounce perfectly over 95% of the time after 50 learning cycles. Details of NETtalk can be found in [11], [49].

B. Time Delay Neural Networks (TDNN)

The time delay neural network model was proposed by Waibel, Hanazawa et.al. [52]. It was constructed primarily for speech learning and recognition. They have listed five properties that a layered feed-forward net must posses in order to be useful for speech recognition. Quoting them "First, it should have multiple layers and sufficient interconnection between units in each of these layers. This is to ensure that the network will have the ability to learn complex non-linear decision surfaces [29]. Second, the network should have the ability to represent relationships between events in time. These events could be spectral coefficients, but might also be the output of higher level feature detectors. Third, the actual features or abstractions learned by the network should be invariant under translation in time. Fourth, the learning procedure should not require precise temporal alignment of the labels that are to be learned. Fifth, the number of weights in the network should be sufficiently small compared to the amount of

training data by extracting regularity." Their claim is that the TDNN they designed satisfies all these five criteria.

Our observation is that while all these criteria must indeed be satisfied the amount of preprocessing used for speech learning in TDNN involving windowing the signal, calculating FFTs and then computing the melscale coefficients is substantial. It would be of interest to explore whether this "preprocessing" could be incorporated within the neural network itself by a suitable choice of network elements and architecture.

1. The TDNN architecture

TDNN uses a modified form of the basic artificial neuron. Apart from the usual linear combiner followed by a non-linearity, Waibel, Hanazawa et.al. introduced delays D_1 through D_N in series with each input, as shown in fig.2.2. Although, not stated explicitly, these delays have been interpreted as $D_i=z^{-i}$ in the discrete time case. Each delay D_i has its own weight w_{j+i}. This way they gave each basic unit the ability to relate the current input to the past history of events, ie. to the inputs sampled previously in time.

The net they used for speech recoginition consisted of 3 layers. At the lowest level, the input to the net comprised a set of 16 normalised melscale spectral coefficients. Speech sampled at 12 kHz was multiplied with a Hamming window and a 256 point FFT was computed. The log energies in each melscale energy band [53] were computed from the power spectrum, where adjacent coefficients in frequency overlapped by one spectral sample and were smoothed by reducing the shared sample by 50 percent [53]. Further data reduction was achieved by collapsing samples adjacent in time which resulted in an overall frame rate of 1 in 10 ms. Fifteen frames of speech were collected centred around the

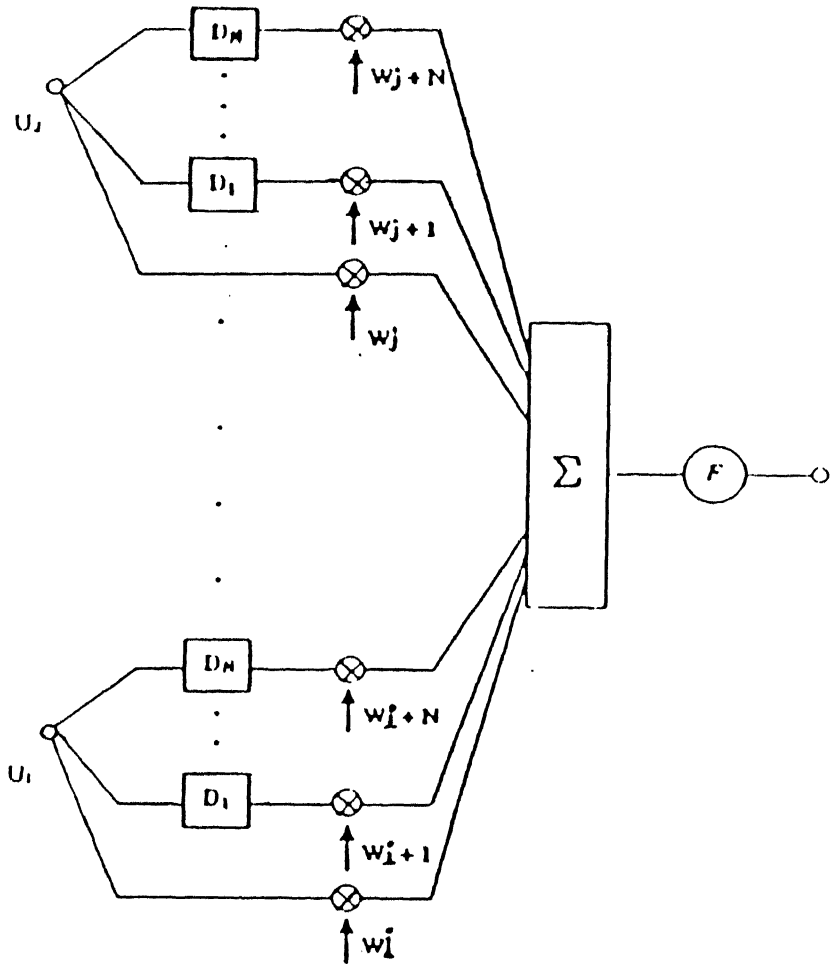


Figure 2.2. A Time-Delay Neural Network (TDNN) unit. (Reproduced from [52])

manually labeled vowel onset and all coefficients of the corresponding input token were normalised so as to lie within +1 and -1.

Each TDNN unit has the ability to encode temporal relationships within the range of the N delays. Short duration local temporal relational features are formed at the lower layer and more complex longer duration features are formed at the higher layer.

Learning is carried out using back-propagation. There are two phases, the first is the forward pass in which an input pattern is applied and the output is calculated, using the current set of weights. The output is compared with the desired target output and the error is calculated. The derivative of the error is propagated back through the network during the second phase and all the weights are adjusted to decrease the error. This is done until the desired output is learned with fair accuracy.

2. Observations and comments on the TDNN

Waibel, Hanazawa et.al. state that in the speech recognition experiment, the recognition rates were above 99.5 percent for 3 speakers in case of TDNN while recognition rates were around 96 to 99 percent in case of the best hidden Markov modeling technique. They state that "better classification rather than better generalization might be the cause of TDNNs better performance."

TDNN is shift-invariant and does not rely on precise alignment or segmentation of input as the delays that have been introduced allow it to learn temporal connections. However, all temporal connections so formed are only upto order N since the maximum delay is D_N. Heuristically, it is more sound to use a general filter model involving use of recursive filters to achieve very long term correlations. This is one reason why we have used a general filter in ALFAN, a

network element described in chapter 3.

One more observation we make is regarding the NETtalk experiment in light of the TDNN. NETtalk used a purely static net which scanned the text using a 5 or 7 character wide window. In some sense this amounts to a model in which a dynamically varying spatial pattern is being presented and by parallely presenting 7 characters simultaneously the temporal variation is being converted into a spatial pattern. Thus, they were able to relate adjacent patterns in time exploiting the fact that adjacent patterns in time could be represented as adjacent patterns in space. An extreme case could be one in which the view window spans the entire word, leaving no room for dynamics at all.

One can argue that the same NETtalk experiment can be conducted using a TDNN and a view-window that spans a single character at a time. This is because, as mentioned earlier, a TDNN uses a ladder of unit delays to store time samples at convenient points in the network. Hence using a maximum delay of 7 units in the network one can possibly generate a context for the input character under consideration at a given time.

C. The "Moving Targets" training paradigm

1. Overview

The "Moving Targets" training algorithm proposed by Rohwer [45] is a simple method for training the dynamical behaviour of a neural network and is applicable to any training problem in discrete-time networks with arbitrary feedback. It is quite similar to the back-propagation paradigm, in that, the gradient of an error measure is used to train the network. The algorithm applies to networks with arbitrary connectivity not just feed-forward, hence it can be used to train recurrent networks for dynamical patterns according to Rohwer.

The key point is that hidden nodes are treated like target nodes with variable training data. Quoting Rohwer "These ***moving targets*** are varied during the minimisation process. Werbos [55] used the term ***moving targets*** to describe the qualitative idea that a network should set itself intermediate objectives, and vary these objectives as information is accumulated on their attainability and their usefulness for achieving overall objectives. The (coincidentally) like-named algorithm presented here [45] can be regarded as a quantitative realization of this qualitative idea."

2. "Moving Targets" training algorithm

Consider a recurrent network with neuron activation values x_{it} taken as the dynamical variables. Suppose x_{it} change with time according to the following law

$$x_{it} = \sum_j w_{ij} f(x_j , t-1) \qquad i > 0 \tag{2.1}$$

$$x_{ot} = \text{constant}$$

Here, w_{ij} are weights from j to i and f is differntiable. The output is given by

$$y_{it} = f(x_{it}) \tag{2.2}$$

In the normal back propagation, a network is divided into input, hidden and target nodes. In the moving targets algorithm, analogous concepts exist but are data dependent instead of depending on the network architecture. Rohwer defines a node-time pair $(i\,t)$ as an ***event***. He distinguishes between 3 disjoint categories of events, namely input I, target T and hidden events H. If X_{it} defines the training data fed to the network as input, then essentially speaking

$$x_{it} = X_{it} \qquad (i\,t) \in I \tag{2.3}$$

The moving targets training method uses an ***activation deficit*** error function given by

$$E_{ad} = \sum_{(i\,t)\in T} \left(x_{it} - X_{it}\right)^2 \tag{2.4}$$

Equation 2 is extended to include hidden events so that

$$E = \frac{1}{2} \sum_{(it) \in T \cup H} \left\{ \sum_j w_{ij} f(X_j, t-1) - X_{it} \right\}^2 \qquad (2.5)$$

This function is minimised to find a suitable set of weights. It can be seen that the weights depend on the moving targets X_{it}. So one needs to compute derivative with respect to moving targets X_{as} alone, where a is the node and s is the time instant and (as) represents an event. The details of the algorithm are given in [45] and presented in fig. 2.3. .

Rohwer found that an output deficit quartic error funciton

$$E = \frac{1}{4} \sum_{(it) \in T} (1.0 + at) \left(y_{it} - Y_{it} \right)^4 \qquad (2.6)$$

a being a small positive constant, yielded better results because the quartic function chooses to enhance larger errors even if they occur infrequently. Further, the temporal weighting encourages the algorithm to focus more on late time errors than on earlier ones. This is in accordance with the causal nature of a real-life system. This also helps in overcoming problems of local minima that may be encountered.

Rohwer tested this algorithm on three different problems — first, in which two inputs, taking 0,1 values, controlled the frequency of an output square wave for the four possible input combinations; second, in which the network learned to generate the time reverse of certain input time sequences; third, in which the network learned to respond to the second of two input pulses separated by 100 time steps, thus demonstrating its capability to grasp distant temporal correlations.

$$E_{ad} = \frac{1}{2} \sum_{(it) \in T} \{x_{it} - X_{it}\}^2 \qquad (a)$$

$$E = \frac{1}{2} \sum_{(t) \in T \cup H} \left\{ \sum_j w_{ij} f(X_{j,t-1}) - X_{it} \right\}^2 \qquad (b)$$

$$\frac{dE}{dw_{ij}} = \frac{\partial E}{\partial w_{ij}} = 0 \qquad (c)$$

$$\frac{dE}{dX_{as}} = \sum_i X_{i,s+1} e_{i,s+1} w_{ia} f'_{as} - X_{as} e_{as} \qquad (d)$$

where

$$X_{it} = \begin{cases} 1 & (it) \in T \cup H \\ 0 & (it) \notin T \cup H \end{cases} \qquad (e)$$

$$e_{it} = \sum_j w_{ij} f(X_{j,t-1}) - X_{it} \qquad (f)$$

$$f'_{it} = \frac{d f(x)}{dx}\Big|_{x=X_{it}} \qquad (g)$$

$$w_{ij} = \sum_k \left(\sum_t X_{it} X_{it} Y_{k,t-1} \right) M_{kj}^{(i)-1} \qquad (h)$$

where

$M^{(a)-1}$ is the inverse of $M^{(a)}$, the correlation matrix given by

$$M_{ij}^{(a)} = \sum_t X_{at} Y_{i,t-1} Y_{j,t-1} \qquad (i)$$

$$\frac{dE}{dX_{as}} = \sum_{ij} \frac{\partial E}{\partial w_{ij}} \frac{dw_{ij}}{dX_{as}} + \frac{\partial E}{\partial X_{as}} = \frac{\partial E}{\partial X_{as}} \qquad (j)$$

Figure 2.3 A summary of the moving targets training algorithm.

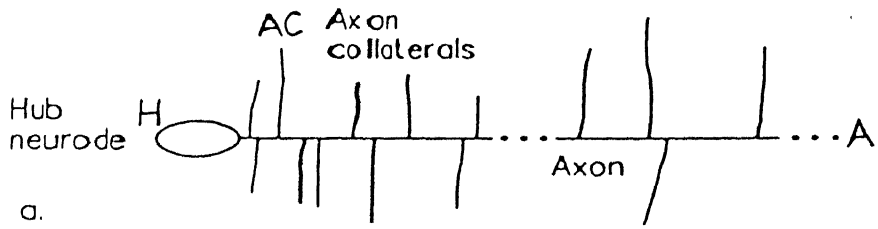
D. The Music-Box associative memory

The music-box associative memory is an old trick and is quite similar to bootstrap loading in computers. It is perhaps the crudest method to build a network that learns spatio-temporal sequences of patterns. Consider a heteroassociative memory that can recall a target pattern when shown an input pattern. The corresponding target is recalled because of an association between the input and the target. Now suppose that the sequence of patterns to be recalled is given as $(P_1,P_2,P_3, \ldots P_N)$. Let P_1 be the target for the given input pattern I, P_2 be the target for P_1 and likewise every succeding P_{i+1} be the target for the previous target pattern P_i. Now if we make an arrangement for the generated target to be fed back as an input after an appropriate delay, then, given the trigger I, the network would recall the sequence $(P_1,P_2,P_3, \ldots P_i,P_{i+1}, \ldots P_N)$.

This is called the **"music-box"** technique because of its similarity to the scheme of generation of a sequence of notes by a music box. Whether this approach amounts to genuine learning of sequences of patterns is debatable. Just consider a sequence $(P_1,P_2,P_3,P_4,P_5,P_2,P_3,P_1,P_4,P_5 \ldots P_N)$. The music box approach would fail in this case because the pattern sub-sequences (P_1) and (P_2,P_3) repeat and the patterns succeeding these subsequences are different at different occurrences. With this approach there are numerous other drawbacks as well [11].

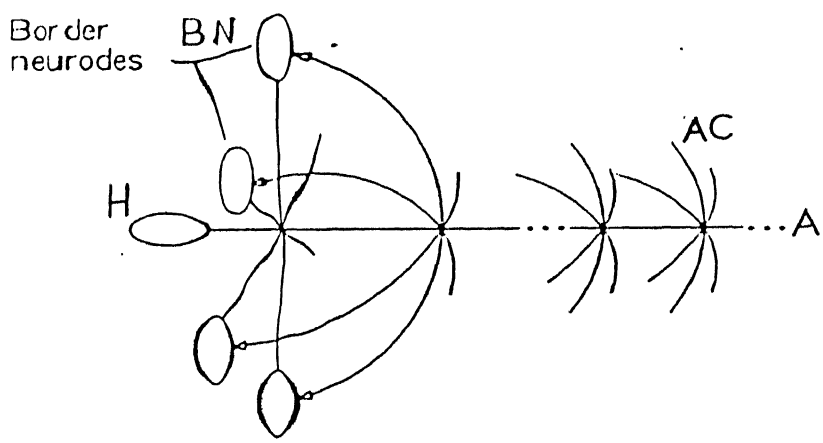
E. Single neurode systems

Caudill and Butler have described some single neurode systems for spatio-temporal pattern learning [11]. An example system, described by Grossberg, using an outstar neurode H [see appendix B for a description of outstar] is shown in fig. 2.4a. A triggering pulse to an outstar neurode travels along the axon A and the axon collaterals AC reaching the ends at different points in time due to



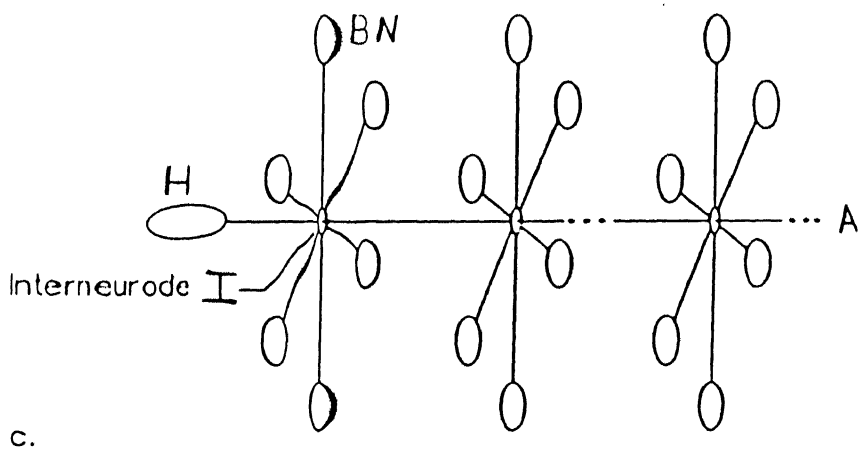
a.

Figure 2.4(a) The simplest outstar capable of learning sequences. (Reproduced from [11])



b.

Figure 2.4 (b) The outstar connected to different synapses of the same grid neurodes makes a more versatile system. (Reproduced from [11])



c.

Figure 2.4(c) A single outstar connected to a series of grids via a set of *interneurons* that pass the signal through. (Reproduced from [11])

different positions and lengths of the axon collaterals. After training, a triggering input would generate a complex spatio-temporal pattern at the far ends of the axon collaterals **AC**. The signals at the ends of these collaterals [marked with arrows in fig. 2.4(a)] can be used appropriately for performing complex tasks like sensory-motor- control of a robot manipulator.

Grossberg, however pointed out that once a trigger is received and a pattern gets generated, there is no way to stop it. Caudill and Butler describe another system shown in fig. 2.4b. It uses an outstar alongwith a set of instar neurodes [see appendix B for a description of instar and outstar]. The outstar is connected to the instar border neurodes **BN** as in the previous case. Here again, the different lengths of axon collaterals **AC** produce different delays at the ends. But each set of signals arrives at a different set of synapses on the instar border neurodes **BN** with different delays, thus creating the same effect as if the hub neurode **H** sequentially activated a series of separate outstars. On proper training and control of arrival times and synaptic efficacies some control can be achieved, via the instar neurodes, on the generated spatio-temporal pattern.

Figure 2.4c shows another structure from Caudill and Butler which uses a series of outstars. A signal from the generating hub **H** is intercepted at varying times by the series of hubs **I** appropriately placed along the axon **A**. These secondary outstar hubs have been called interneurodes because of their functional similarity to the biological interneurons. The instar border neurodes **BN** in the principal and secondary outstars can communicate with other similar networks, hence a variety of complex spatio-temporal patterns can be generated upon enforcing suitable control and training. As mentioned earlier, some such appropriately generated patterns can be used in complex tasks requiring co-ordination, such as, control of robot manipulators.

In all these networks appropriate spacing between neurodes and proper choice of lengths for the axon collaterals are very crucial points. One has to design suitable training procedures to identify the correct values for the delays. This is definitely not an easy job nor are we aware of any systematic enquiry or empirical research that may have been conducted in this connection. Further, all these networks are only to generate spatio-temporal patterns and not for recognising such patterns. Hence their utility is limited since many conventional methods already exist for generating spatio-temporal patterns.

F. The outstar avalanche system

Outstar avalanche network modifies the basic neuron used. Normally the activity of an outstar neurode is assumed to die instantly. However, in this network an exponential decay for the neurode output is assumed. The activation fall off is controlled by a single decay parameter called activation decay constant. The larger the decay constant the shorter is the response duration.

The avalanche network consists of three layers — an input, a middle and an outer layer. The function of the input layer is to transmit the incoming trigger signal to all the neurodes of the middle layer. Hence, functionally, each input neurode is the hub of an outstar [see appendix B for a description of outstar].

The output layer generates the desired patterns and each neurode in this layer is the focus of an instar beginning on neurodes in the previous layer. The output neurodes each have an external input through which the training patterns are impressed during the learning phase. The middle layer neurodes receive inputs from other neurodes in the same layer, as well as, from those in the input layer.

Once trained, the avalanche network works as follows. The middle or avalanche layer neurodes fire only upon receiving a stimulus from the currently

active neurode and if the previously firing neurodes in the sequence are still at least partially active. In other words the exponentially decaying activation introduces a memory into the system and is capable of exhibiting long term correlations. The succeeding neurodes are triggered only if the correct combination of stimuli is received with proper temporal alignment. Suppose the middle layer neurodes are excited in the wrong order or accidentally stimulated with noise, any resulting spurious activity in the layer extinguishes rapidly. Further, for the generation of more complex patterns we can make it necessary that more than one neurode is used to trigger the process.

Training this network is not difficult. First of all the avalanche layer is activated by presenting the desired input pattern to the network. In correspondence with the incoming sequence, the target spatio-temporal pattern is also presented to the network through the input terminals of the output neurodes. The neo-Hebbian learning procedure used in case of outstar learning is employed in this case as well. With repeated presentations the avalanche network learns the desired sequence of patterns. The network can learn a multitude of sequences and the generation of complex patterns can be controlled using the many available inputs to it with appropriate training.

While all these outstar based networks can generate spatio-temporal patterns after appropriate training, it is probably inadequate for spatio-temporal pattern recognition. Robert Hecht-nielsen suggested that a Kohonen learning scheme can be incorporated into the avalanche layer of the network. An unknown sequence of patterns can be compared with a learned sequence of patterns using competitive learning strategy used in Kohonen's feature map generating networks. A good discussion of this is given in [11].

IV. Some other approaches to spatio-temporal pattern analysis – Self-organising structures for temporal pattern analysis

Self-organising neural systems have been extensively used for spatial pattern recognition tasks like handwritten character recognition. Notable among these networks are the ART1 and ART2 proposed by Grossberg [8],[9], the Cognitron [15], the Neocognitron [16], and their variants proposed by Fukushima and others, and lastly, Kohonen's self-organising feature maps [27].

Some researchers have considered these networks as possible means of recognising temporal patterns like speech. One of the interesting applications is of speech segmentation and phoneme recognition. We have hitherto seen how the HMM technique is used for state estimation to carry out speech segmentaion. HMM requires that a model be specified for this task. HMM uses complex algorithms to estimate the state of the system on the basis of which speech segmentation is performed [42], [44]. In contrast, Self-organising neural networks, by virtue of their ability to learn and classify without supervision, do not require any information, apriori. This allows us to appropriately use these networks to perform speech segmentation, specifically to detect phoneme boundaries in a continuous speech utterance. Self-organising networks, upon adapting suitably to a training set, should be in a position to detect a change in pattern when one phoneme ends and the next begins, without any supervision .

Assume that N consecutive samples or melscale coefficients [53] of a speech sequence are fed to an ART2 or Neocognitron [see appendix C for a description of Neocognitron]. A process of self-organistation will take place with the repeated application of the input. Suppose the input speech consists of a sequence of elementary utterances – phonemes or other such entities. If we assume that self organisation process assigns a single gnostic cell to each elementary utterance

then after training has sufficiently progressed we could expect that upon impressing an elementary utterance the corresponding gnostic cell would fire. As soon as one input pattern leads to another (a difference that is percieved by the network upon self-organisation) the gnostic cells would also switch correspondingly, thus performing a segmentation of the input sequence. The process of self-organisation excludes human intervention or help and makes it a fully automatic procedure for performing segmentation of the input sequence. The input can be presented as a set of N samples seen through a window of width N which moves $M < N$ steps every time instant. The necessary preprocessing can also be done to present a set of melscale or short-time Fourier transform coefficients at periodic intervals. A shift-invariant network like neocognitron or its derivatives would be very effective in such an event because its ability to recognise shifted versions of a pattern would enable it to very effectively recognise and segment non-stationary signals like speech.

We have tried using the neocognitron for the speech segmentation experiment. A program for simulating the neocognitron has been written. The number of layers and the number of neurons per layer are limited only by the computer's memory. A dynamic data structure has been used to store the neuron activation values. We simulated the system and trained the network to learn some static patterns. While the network did acquire the ability to fire the gnostic cells, it failed to distinguish between the two input pictures that were fed to it. This was the case for several pairs of pictures fed to it. The neocognitron seemed to be very highly sensitive to learning parameters, various thresholds and initial network weights. While self-organisation did take place the network did not acquire the ability to distinguish between two different patterns. After several efforts to overcome these limitations we had to abandon this line of our investigations into

temporal pattern recognition with neural networks.

V. Motivation and heuristics for the conception of a Linear Filter Artificial Neuron System

This section serves as a prelude to chapter 3 in which we describe a network called ALFANS, standing for A Linear Filter Artificial Neuron System. It is comprised of artificial neurons each of which is followed by a linear filter. Here, we discuss the issues and ideas that led us to construct ALFANS. Researchers have hitherto considered neural elements similar to ALFAN for pattern recognition tasks but we are unaware of any possible systematic study of such elements for spatio-temporal pattern recognition.

ALFANS uses linear filters which facilitate the introduction of finite or infinite duration memory into the system. Careful inspection of fig.2.5 would reveal that ALFAN is similar to Grossberg's avalanche outstar and to the TDNN element for particular cases of the linear filter. However, it was conceived independently and in complete ignorance of these models only to be surprised later by the striking similarity between the elements of the three systems.

It was long felt that the basic neuron structure was inadequate for spatio-temporal pattern learning and establishing connections in time. The first modification we made was to incorporate delays $z^{-\tau_i}$ in series with each input x_i, where z^{-i} denotes a unit delay (refer to fig.2.6). The network was a simple feed-forward system and learning was to be via the back-propagation method. The weights w_{ij} and the delays τ_i were to be learned. However, the learning algorithm seemed difficult to formulate in this case. Refering to fig. 2.6 we can see that this model is more economical than the TDNN scheme but requires a training procedure to modify the parameters τ_i alongwith the weights.

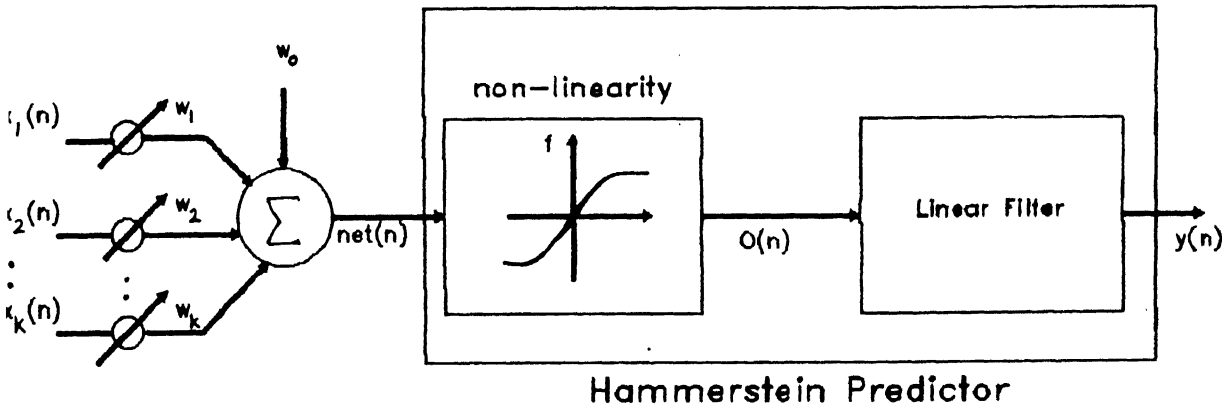


Figure 2.5 A Linear Filter Artificial Neuron

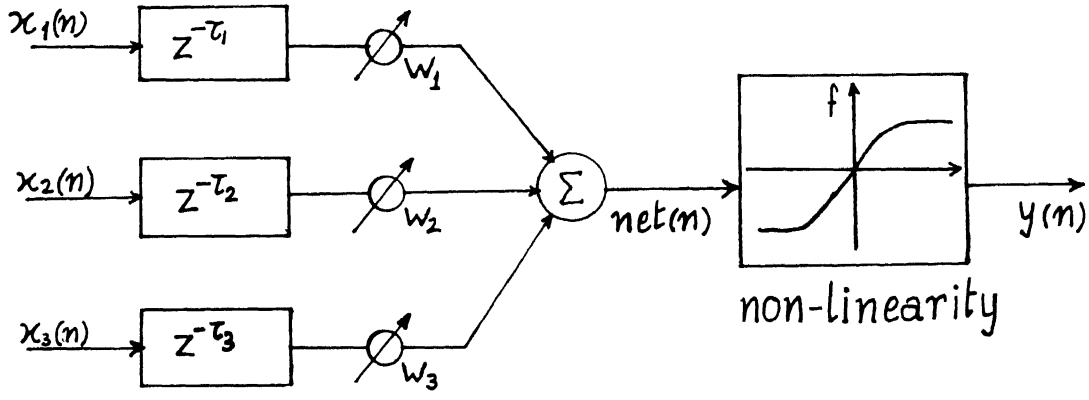


Figure 2.6 A Time-Delay neural unit

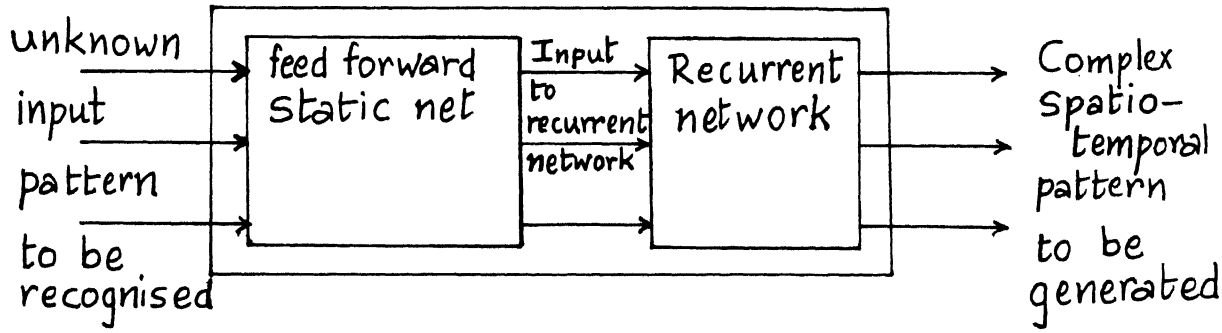


Figure 2.7 A scheme employing a static feedforward network as a pre-processor for a recurrent network for spatio-temporal pattern recognition.

One idea we had was of implementing the delays $z^{-\tau_i}$ using neural elements alone. This can be achieved by constructing neural analogs of delay flip-flops using a simple recurrent network. This is simply done by making NAND gate equivalents of neural networks and then constructing a delay flip-flop. It was felt that perhaps a recurrent net itself could be designed using smaller flip-flop like networks and "neural gating schemes" to allow the system to learn a multitude of patterns and generate them upon receiving suitable input patterns.to trigger. To learn binary temporal sequences of length N it can be shown that (N+1) sequential neural structures are needed and 2^N possible sequences exist. Thus, the recurrent neural system would be just a counter which receives inputs and changes its state deterministically depending on its current state and the input at that instant.

Making the above mentioned network recognise noisy and corrupted versions of a time pattern would require that the input sequences be tabulated and the network be constructed using the usual Boolean logic techniques. In other words, suppose there are P patterns to be learnt; we tabulate the P sequences each of length N and construct the network taking all other sequences as don't cares or as forbidden sequences. Then we use a static feed-forward back-propagation net to generate the triggering and control inputs to the recurrent network, on receiving a corrupted sequence of input patterns from the world outside. The scheme is depicted in the block diagram of fig.2.7. In lieu of the feed-forward net one can use any other scheme like the ART, the Cognitron, or Kohonen's net and input to the recurrent network would then be the output of the gnostic cells in the recognition layer of these nets.

This network is difficult and very tedious to implement. So we went back to the original feed-forward network with delays $z^{-\tau_i}$ and decided to put the delays at the output of the neurons instead of putting them on the input side. Finally, we

replaced the delays with general linear filters of first and second order. The linearity of the filters allowed us to use a back-propagation based learning algorithm. The linear filter artificial neuron system is presented in chapter 3 along with the derivation of the learning algorithm. The results of our simulation are presented in the chapter 4.

CHAPTER 3

A Linear Filter Artificial Neuron System For Spatio-Temporal Pattern Recognition

I. Introduction

We have already examined the relevance of neural networks in the context of temporal pattern recognition in chapter 2. We have also reviewed some recent work done in this context.

In this chapter we present a new back-propagation algorithm for training a class of neural networks for spatio-temporal pattern recognition. The basic unit in these networks is **a linear filter artificial neuron (ALFAN)** which is basically, an artificial neuron followed by a linear time-varying or time-invariant causal filter. The filter could be of recursive or non-recursive type. In this sense, essentially, ALFAN is a linear combiner followed by a Hammerstein predictor [10]. A Hammerstein predictor is a non-linearity followed by a linear filter.

Narendra and Parthasarathy [40] have talked of a network quite similar to ALFAN system in the context of system identification. We present the algorithm in this chapter while the results of our simulation will be presented in chapter 4.

II. ALFAN structure and problem definition

Our interest is in recognising spatio-temporal patterns using a supervised learning neural system. One of the promising candidates to achieve the stated objective with much convenience is a layered feed-forward dynamic network. By using the term feed-forward in the context of a dynamical system we mean, a

layered network in which the output of a layer never returns to its preceeding layers during the recognition phase, either directly or indirectly. Infact, in ALFAN system the output is never fedback to the input of the same neuron or even to the inputs of other neurons in the same layer. The structure of ALFAN element is as shown in fig. 3.1. The following equations describe the input-output relationship for ALFAN.

$$net(n) = \sum_{k=1}^{K} w_k x_k(n) + w_0 \qquad (3.1)$$

$$O(n) = f[Net(n)] \qquad (3.2)$$

$$y(n) = h(n) * O(n)$$

$$= \sum_{m=0}^{N-1} h(n,m;a)\, f[net(m)] \qquad (3.3)$$

In the above system of equations, h(n,m;a) is a causal time-varying or time-invariant filter with a parameter vector a. Here, h(n,m;a) is the response at time n to an impulse applied at time m. A four layer feed-forward net has the structure shown in fig. 3.2. The input layer $\acute{L}$ is a fictitious layer and does not have any ALFAN elements in it. The layer $\acute{K}$ is the output layer and generates an approximation to the desired target patterns.

We start with a set of inputs to the network. Each input is an N length sequence of real vectors. A set of P N-length sequences of real vectors constitute the desired targets to be generated when recognition is performed on the input set. When the input sequence is presented in time to the input layer the non-linear network must produce at the output, one of the P target sequences or a

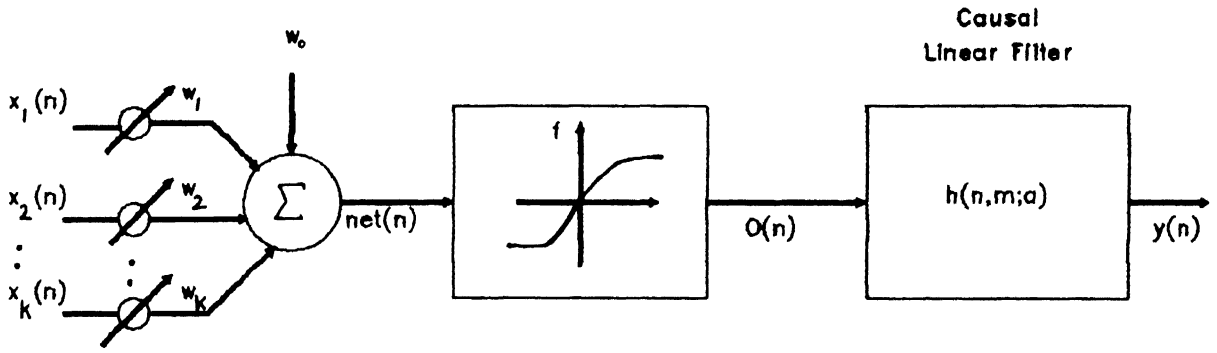


Figure 3.1 A Linear Filter Artificial Neuron

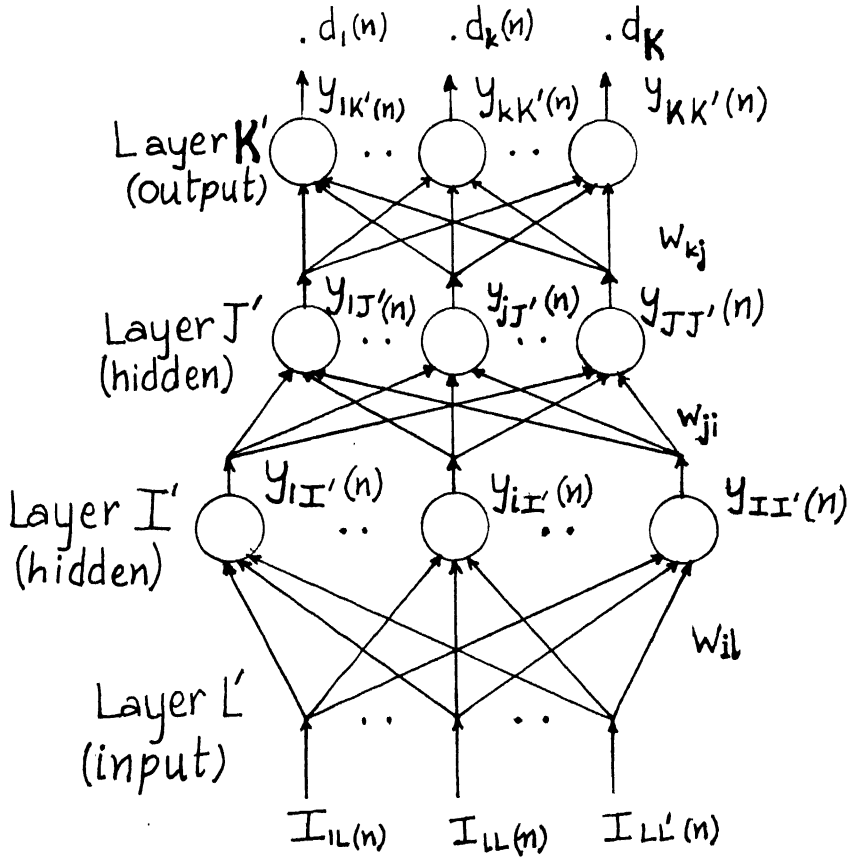


Figure 3.2 A Linear Filter Artificial Neuron System containing four layers.

close approximation to it, and this must be simultaneous with the input. The linear filter with finite or infinite memory is used to aid in the task of generating the output time sequence while the non-linear neuron is used to perform a suitable partitioning of the input and recognition spaces to select the desired output pattern.

Learning process is initiated by a suitable choice of initial neuron weights and filter parameters. The initial neuron weights are chosen at random and the filter parameters too chosen randomly, conform to some suitably specified restrictions that ensure the stability of the linear filter. Now suppose the network has K neurons in the output layer $\acute{K}$ and L neurons in the input layer $\acute{L}$. An exemplar input time sequence of length N is denoted by the vector $I_{\acute{L}}^{p}(n) \in \Re^{L}$, n running from 0 to N-1. The superscript p denotes the number of the input or target pattern and is an integer from 1 to P. The desired output sequence or target is a vector sequence $D_{\acute{K}}^{p}(n) \in \Re^{K}$. More explicitly,

$$D_{\acute{K}}^{p}(n) = [\, d^{p}_{1\acute{K}}(n) \cdots d^{p}_{K\acute{K}}(n) \,].$$

The output generated by layer $\acute{K}$ is

$$Y^{p}_{\acute{K}}(n) = [\, y^{p}_{1\acute{K}}(n) \cdots y^{p}_{K\acute{K}}(n) \,].$$

Our objective is to make signal $Y^{p}_{\acute{K}}$ the closest possible to the target $D^{p}_{\acute{K}}$ and farthest possible from other targets $D^{\acute{p}}_{\acute{K}}$ where $\acute{p}$ is not equal to p.

The error minimisation between the target and the generated sequence is done using a least squares criterion which is used to modify the neural weights and filter parameters at the end of each learning cycle or at the conclusion of alternate learning cycles. We chose the latter option in which we modify the weights keeping the filter parameters constant and then modify the filter parameters using the new set of weights, in the next learning cycle.

To carry out supervised learning we define the following quadratic error function

$$E=\sum_{p=1}^{P}\sum_{n=0}^{N-1}\sum_{k=1}^{K}\left(d^{p}_{k\acute{K}}(n)-y^{p}_{k\acute{K}}(n)\right)^{2}\frac{2(n+1)}{(N+1)} \qquad (3.4)$$

where P is the number of exemplar patterns and N is the length of each of the time sequences. The weight factor $\frac{2(n+1)}{(N+1)}$ is used to give more importance to errors in output samples occuring at later time instants. Some such scheme is appropriate as the system is causal and cannot be expected to recognise or regenerate patterns before they are applied. Further, its ability to classify or predict increases as more time samples arrive at the input. Our aim now would be to minimise E by executing a training procedure, modifying the weights of the network and parameters of the filter suitably.

In the following we shall adopt the simpler notation of single subscripts that neural net researchers commonly employ. We represent the elements of the target pattern vector as $d^{p}_{k}(n)$ and the output values of layer $\acute{K}$ as $y^{p}_{k}(n)$, while the output of the inner layer $\acute{J}$ will be written as $y^{p}_{j}(n)$ and so on. Hence, we can write (3.4) as

$$E=\sum_{p=1}^{P}\sum_{n=0}^{N-1}\sum_{k=1}^{K}\left(d^{p}_{k}(n)-y^{p}_{k}(n)\right)^{2}\frac{2(n+1)}{(N+1)} \qquad (3.5)$$

As

$$dE=\frac{\partial E}{\partial \mathbf{w}}d\mathbf{w}+\frac{\partial E}{\partial \mathbf{a}}d\mathbf{a}$$

where $\mathbf{w}$ is the vector of all weights and $\mathbf{a}$ is the vector of all filter parameters, by forcing both partial derivatives above to zero, a point of local minimum for the error E in the network's parameter space can be reached.

The recognition process, also called the testing phase, is fairly straight

forward. We apply a vector time pattern [$I_{1L}'(n) \cdots I_{1L}'(n) \cdots I_{LL}'(n)$] for n= 0 to N-1, just as we apply a vector time signal to a multi-input filter. The neural net is updated layer by layer starting from the input to the output, for succesive time instants, from n= 0 to N-1. The output layer thus generates in time, a close approximation to the desired temporal pattern. As we have seen, the training procedure for the neural network is far more complex and time consuming. However, it is executed only when a new pattern is to be taught to the network. In contrast the recognition process is very fast and simple and is used to recognise patterns when the network is put to use.

III. ALFAN training algorithm

A. Learning in the output layer

Let the change in weight w_{kj} on learning iteration t be Δw^t_{kj} so that

$$w_{kj}^{t+1} = w^t_{kj} + \Delta w^t_{kj} \tag{3.6}$$

then by choosing Δw^t_{kj} to be proportional to $-\frac{\partial E}{\partial w_{kj}}\Big|_t$,

$$\Delta w^t_{kj} = -\eta \frac{\partial E}{\partial w_{kj}}\Big|_t \tag{3.7}$$

the error E can be reduced from its value corresponding to the weights w^t_{kj}. The rate of learning η is a positive constant and could be different for different layers. Similarly,

$$a_k^{t+1} = a_k^t + \Delta a_k^t$$

$$\Delta a_k^t = -\eta_a \frac{\partial E}{\partial a_k}\Big|_t \tag{3.8}$$

1. Modifying the weights

Using (7) we have

$$\Delta w^t_{kj} = -\eta \frac{\partial}{\partial w_{kj}} \left\{ \sum_{p=1}^{P} \sum_{n=0}^{N-1} \sum_{k=1}^{K} \left(d^p{}_k(n) - y^p{}_k(n) \right)^2 \frac{2(n+1)}{(N+1)} \right\}$$

$$\Delta w^{t}_{kj} = \sum_{p} \Delta w_{kj}^{t,p} \tag{3.9}$$

where $\Delta w_{kj}^{t,p}$ is the change associated with pattern p after learning cycle t.

Noting that

$$y^{p}_{k}(n) = \sum_{m=0}^{N-1} h_k(n,m;a_k) f[net^{p}_{k}(m)] \tag{3.10.1}$$

$$net^{p}_{k}(m) = \sum_{j} w_{kj} y_j(m) \tag{3.10.2}$$

and on omitting superscripts p and t we have

$$\Delta w_{kj} = -2\eta \frac{\partial}{\partial w_{kj}} \left\{ \sum_{n=0}^{N-1} \sum_{k} \left(d_k(n) - \sum_{m=0}^{N-1} h_k(n,m;a_k) \; f[\sum_{j} w_{kj} y_j(m)] \right)^2 \frac{(n+1)}{(N+1)} \right\}$$

$$= 4\eta \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} \frac{(n+1)}{(N+1)} [d_k(n) - y_k(n)] \; f'[net_k(m)] \; h_k(n,m;a_k) \; y_j(m) \tag{3.11}$$

Here we have assumed that $h_k(n,m;a_k)$ is a causal linear filter with parameter vector a_k. In other words

$$h_k(n,m;a_k) = 0 \quad \text{for} \quad n < m \tag{3.12}$$

Define

$$\delta_k(m) = 4 \sum_{n=0}^{N-1} \frac{(n+1)}{(N+1)} [d_k(n) - y_k(n)] \; f'[net_k(m)] \; h_k(n,m;a_k) \tag{3.13}$$

so that

$$\Delta w_{kj} = \eta \sum_{m=0}^{N-1} \delta_k(m) y_j(m) \tag{3.14}$$

2. Modifying the filter parameters

Proceeding exactly as we did before we have

$$\Delta a_k{}^{t} = \sum_{p} \Delta a_k{}^{t,p} \tag{3.15}$$

For pattern p, after learning cycle t,

$$\Delta a_k = 4\eta_a \sum_{n=0}^{N-1} \frac{(n+1)}{(N+1)} [d_k(n) - y_k(n)] \sum_{m=0}^{N-1} f[net_k(m)] \frac{\partial h_k(n,m;a_k)}{\partial a_k} \qquad (3.16)$$

$$= 4\eta_a \sum_{n=0}^{N-1} \frac{(n+1)}{(N+1)} [d_k(n) - y_k(n)] v_{a_k}(n) \qquad (3.17)$$

where

$$v_{a_k}(n) = \sum_{m=0}^{N-1} f[net_k(m)] \frac{\partial h_k(n,m;a_k)}{\partial a_k} \qquad (3.18)$$

$$= \frac{\partial h_k(n,m;a_k)}{\partial a_k} * f[net_k(n)]$$

where * denotes superposition operation. The structures shown in figures 3.3a and 3.3b may be used to obtain the increments to the filter parameters of the outer and inner layers respectively. We analyse the conditions for stability in sub-section C.

B. Learning in the inner layers

Learning in the inner layers which are also referred to as the hidden layers, is carried out in the same manner as in conventional Back-Propagation algorithm for static patterns except that the algebra is more complex. The parameters of filters in the hidden layers must also be adapted and they, as well, depend on the error that is propagated inwards from the output layer.

1. Modifying the weights

The weights w_{ji} from hidden layer $\acute{I}$ to hidden layer $\acute{J}$ are modified according to the following scheme employing the propagation variable $\delta_k(m)$.

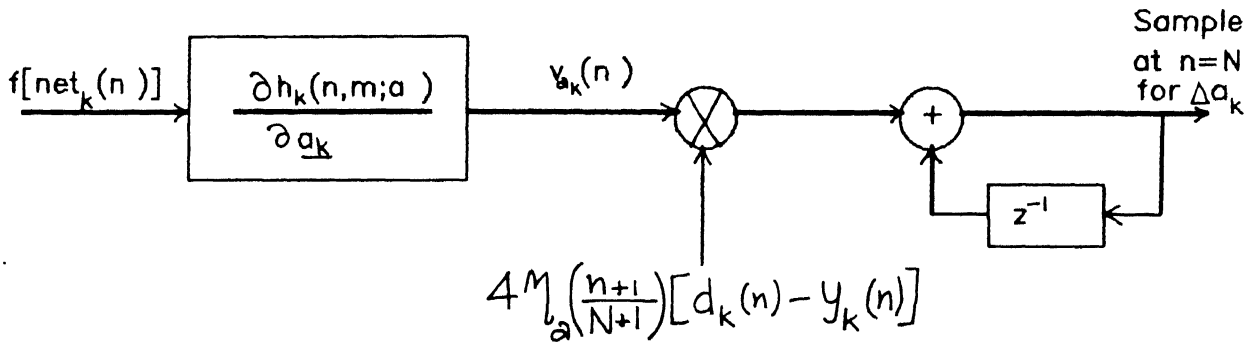


Figure 3.3(a) Structure to obtain parameter increments Δa_k of the outer layer K.

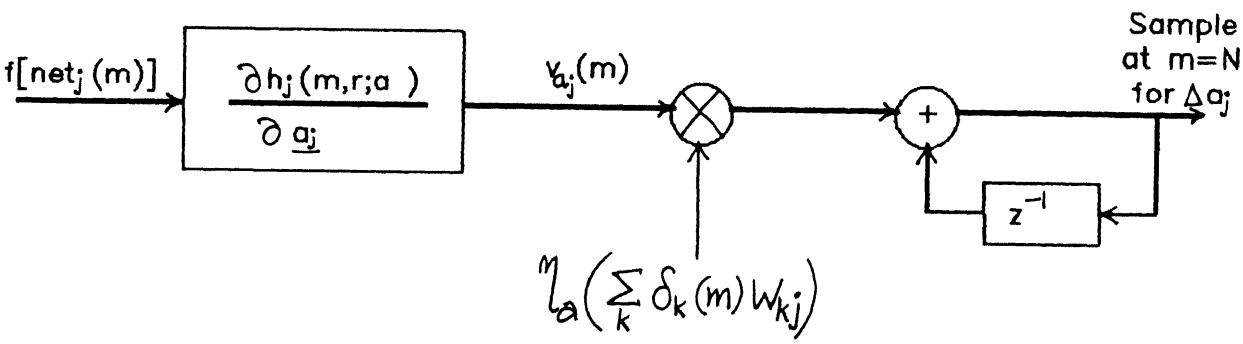


Figure 3.3(b) Structure to obtain parameter increments Δa_j for the hidden layer J.

We have

$$\Delta w_{ji} = -2\eta \frac{\partial}{\partial w_{ji}} \left\{ \sum_{n=0}^{N-1} \sum_{k} \left(d_k(n) - \sum_{m=0}^{N-1} h_k(n,m;a_k)\, f[net_k(m)] \right)^2 \frac{(n+1)}{(N+1)} \right\} \tag{3.19}$$

$$= 4\eta \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} \sum_{k} \frac{(n+1)}{(N+1)} [d_k(n) - y_k(n)]\, f'[net_k(m)]\, h_k(n,m;a_k) \frac{\partial net_k(m)}{\partial w_{ji}} \tag{3.20}$$

$$= 4\eta \sum_{m=0}^{N-1} \left\{ \sum_{k} \left(\sum_{n=0}^{N-1} \frac{(n+1)}{(N+1)} [d_k(n) - y_k(n)]\, f'[net_k(m)]\, h_k(n,m;a_k) \right) w_{kj} \right\} \cdot$$

$$\sum_{r=0}^{N-1} h_j(m,r;a_j)\, f'[net_j(r)]\, y_i(r) \tag{3.21}$$

From the definition of $\delta_k(m)$ in (13) we can write

$$\Delta w_{ji} = \eta \sum_{r=0}^{N-1} \sum_{m=0}^{N-1} \left(\sum_{k} \delta_k(m) w_{kj} \right) h_j(m,r;a_j)\, f'[net_j(r)\, y_i(r) \tag{3.22}$$

Furthermore, we define

$$\delta_j(r) = \sum_{m=0}^{N-1} \left(\sum_{k} \delta_k(m) w_{kj} \right) h_j(m,r;a_j)\, f'[net_j(r) \tag{3.23}$$

so that we have

$$\Delta w_{ji} = \eta \sum_{r=0}^{N-1} \delta_j(r)\, y_i(r) \tag{3.24}$$

which identical, in form, to the expression for Δw_{kj} .

2. Modifying the filter parameters

The filter following the non-linearity in a layer j neuron has been represented as $h_j(n;a_j)$. The increments $\Delta a_j t$ are computed next.

We have

$$\Delta a_j^t = -\eta_a \frac{\partial E}{\partial a_j}\Big|_t$$

$$= 4\eta_a \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} \sum_k \frac{(n+1)}{(N+1)} [d_k(n) - y_k(n)] f'[net_k(m)] h_k(n,m;a_k) w_{kj} \frac{\partial y_j(m)}{\partial a_j} \qquad (3.25)$$

$$= 4\eta_a \sum_{m=0}^{N-1} \left\{ \sum_k \left(\sum_{n=0}^{N-1} \frac{(n+1)}{(N+1)} [d_k(n) - y_k(n)] f'[net_k(m)] h_k(n,m;a_k) \right) w_{kj} \right\} \cdot$$

$$\sum_{r=0}^{N-1} \frac{\partial h_j(m,r;a_j)}{\partial a_j} f[net_j(r)] \qquad (3.26)$$

$$= \eta_a \sum_{m=0}^{N-1} \left\{ \sum_k \delta_k(m)\, w_{kj} \right\} v_{a_j}(m) \qquad (3.27)$$

where

$$v_{a_j}(m) = \frac{\partial h_j(m,r;a_j)}{\partial a_j} * f[net_j(m)] \qquad (3.28)$$

On similar lines we can proceed for the other inner layers using the propogation variable $\delta(\cdot)$. So we use $\delta_j(r)$ to train layer j and so on.

IV. Choice of filter and conditions for stability

ALFAN model makes it necessary that the filter be linear and causal. Three different linear time-invariant IIR filters were used for simulation purposes, namely,

(a) A first order filter with impulse response a^n with $|a| < 1$

(b) A second order resonator with imaginary poles and impulse response $\cos(\omega n)$

(c) A second order section whose poles can lie within a certain region of the unit circle.

With every filter $h(n;a)$ a set of filters $\frac{\partial h(n;a)}{\partial a}$ are associated. We must ensure the stability of all these filters as well.

A. Decaying exponential impulse response

1. The DERIVATIVE filter

This is the simplest of all non-trivial cases associated with IIR filters.

$$h(n;a) = a^n \quad ; \qquad |a| < 1 \quad \text{for stability.} \tag{3.29}$$

The Z- transform of h is

$$H(z) = \frac{1}{(1-az^{-1})} \quad ; \quad |z| \leq 1 \tag{3.30}$$

so that

$$y(n) = a\, y(n-1) + f[net(n)] \tag{3.31}$$

Now

$$v_a(n) = \frac{\partial h(n;a)}{\partial a} * f[net(n)] \tag{3.32}$$

$$\frac{\partial h(n;a)}{\partial a} = n\, a^{n-1}$$

$$= \frac{n}{a}\, a^n \tag{3.33}$$

We make use of the following property of the Z-transform to derive the filter structure for $\frac{\partial h}{\partial a}$:

$$-z \frac{dH(z)}{dz} = Z\{n\, h(n)\} \tag{3.34}$$

From (33) and (34) we have

$$\frac{\partial h(n;a)}{\partial a} = Z^{-1}\left\{ -\frac{z}{a} \frac{d}{dz}\left(\frac{z}{z-a} \right) \right\} \tag{3.35}$$

If we define

$$f[net(n)] = x(n)$$

then

$$\frac{V_a(z)}{X(z)} = -\frac{z}{a} \frac{d}{dz}\left(\frac{z}{z-a} \right)$$

$$= \frac{z^{-1}}{1 - 2a\, z^{-1} + a^2\, z^{-2}} \tag{3.36}$$

Hence

$$v_a(n) = 2a\, v_a(n-1) - a^2\, v_a(n-2) + f[net(n-1)] \tag{3.37}$$

The filter $v_a(n)$ is shown in fig. 3.4.

2. Conditions for the stability of the DERIVATIVE filter $\frac{V_a(z)}{X(z)}$

To check the stability of a discrete time filter we use Jury's Test which states the conditions under which a filter

$$H(z) = \frac{N(z)}{D(z)} \quad ; \qquad \text{with} \quad D(z) = \sum_{i=0}^{N} b_i \, z^{n-i}$$

is stable. Appendix A provides the pertinent details.

Clearly, the first two conditions in Jury's Test, namely, $D(-1) > 0$ and $D(1) > 0$ are trivially satisfied for all real a. Further, on writing down the table (see appendix A) we get $|a| < 1$, which is same as the condition on h(n;a). Hence, while training we can modify parameter a associated with each filter so long as it is within the limits prescribed above. If $|a^{t+1}| > 1$ then we freeze it at a value close to 1, say, 0.88, thus ensuring the stability of the filters assuming reasonable truncation errors.

B. Co-sinusoid impulse response

In this case our filter is a resonator

$$h(n;\omega) = \cos\omega n \tag{3.38}$$

Correspondingly,

$$H(z) = \frac{z\,[\,z - \cos\omega\,]}{z^2 - 2z\cos\omega + 1} \tag{3.39}$$

hence

$$y(n) = 2y(n-1)\cos\omega - y(n-2) + f[net(n)] - f[net(n-1)]\cos\omega \tag{3.40}$$

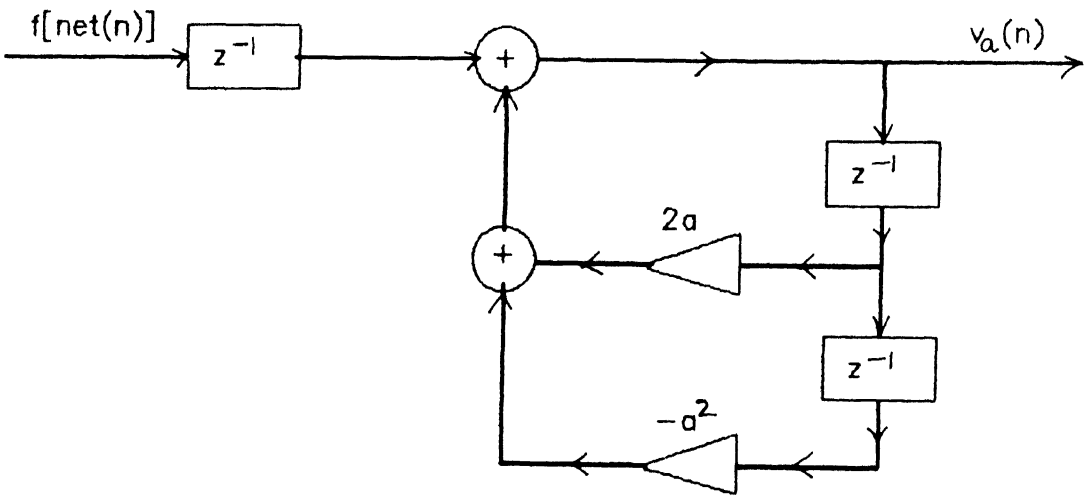


Figure 3.4 The DERIVATIVE filter for case (a). [See eqn. 3.37]

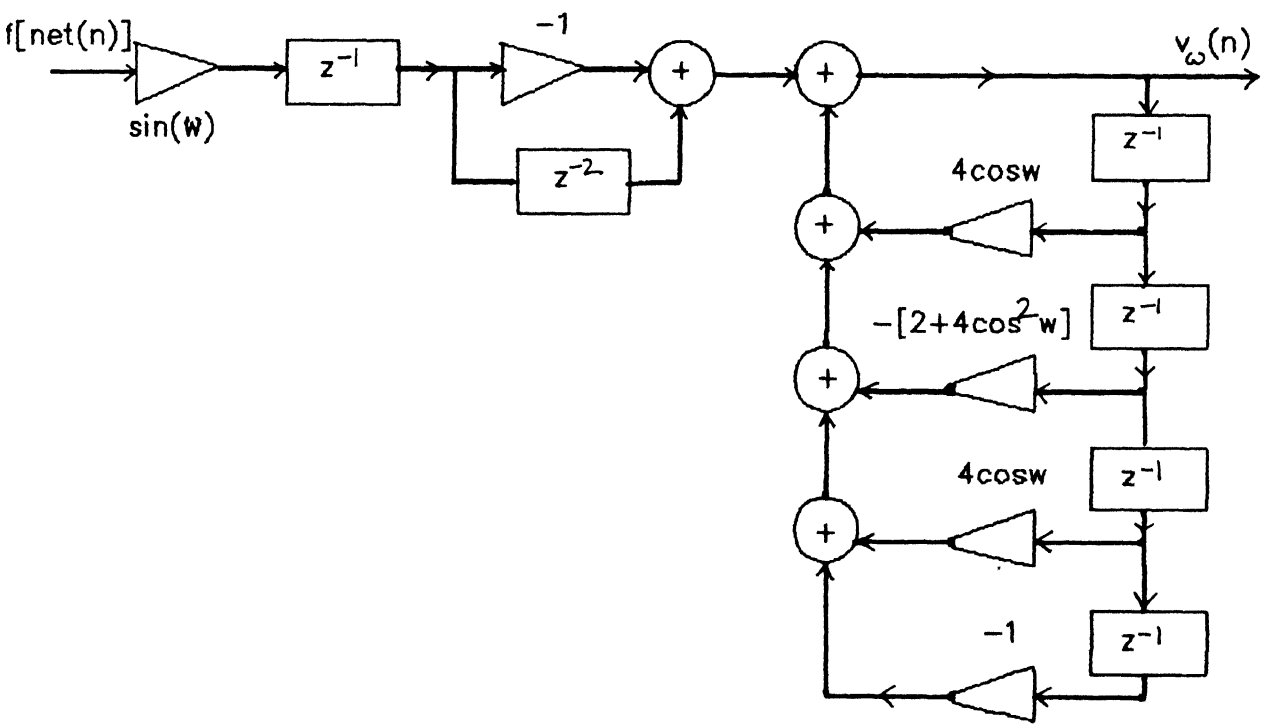


Figure 3.5 The DERIVATIVE filter for case (b). [See eqn. 3.44]

. The DERIVATIVE filter $\frac{V_\omega(z)}{X(z)}$

The Derivative filter must have an impulse response given by

$$\frac{\partial h(n;\omega)}{\partial \omega} = -n \sin\omega n \qquad (3.41)$$

$$= Z^{-1}\left(z \frac{d}{dz}\left(\frac{z \sin\omega}{z^2 - 2z\cos\omega + 1} \right) \right)$$

Computing the derivative we have

$$\frac{V_\omega(z)}{X(z)} = \frac{\sin\omega\,(z - z^3)}{(z^2 - 2z\cos\omega + 1)^2} \qquad (3.42)$$

$$= \frac{\sin\omega\,(z^{-3} - z^{-1})}{1 - 4z^{-1}\cos\omega + z^{-2}(4\cos^2\omega + 2) - 4z^{-3}\cos\omega + z^{-4}} \qquad (3.43)$$

Hence, we implement the filter using the following difference equation

$$v_\omega(n) = 4v_\omega(n-1)\cos\omega - v_\omega(n-2)[4\cos^2\omega + 2] + 4v_\omega(n-3)\cos\omega - v_\omega(n-4)$$

$$+ \sin\omega\,\{\, f[net(n-3)] - f[net(n-1)]\,\} \qquad (3.44)$$

The filter for $v_\omega(n)$ is shown in fig. 3.5. Clearly, the poles of this filter lie on the unit circle and the filter is BIBO stable for any value of ω.

C. Decaying co-sinusoid impulse response

This is the most general of the three cases listed under section III. This is not only more useful but also more interesting than the other two cases considered hitherto. Here we have

$$h(n;a,\omega) = a^n \cos\omega n$$

Correspondingly, the transfer function associated with it is given by

$$H(z) = \frac{z\,[\,z - a\cos\omega\,]}{z^2 - 2z\,a\cos\omega + a^2} \qquad (3.45)$$

From the impulse response it is patent that for stability $|a| \leq 1$. Now we have the filter parameter vector a consisting of two scalars $[a,\omega]$. We have the task of adapting both the parameters during the learning phase. Hence, we obtain filters

corresponding to two different derivatives, namely, with respect to a and with respect to ω. We utilise the filters corresponding to these while computing the increments Δa^t and $\Delta \omega^t$.

1. The DERIVATIVE filter $\frac{V_a(z)}{X(z)}$

First we proceed with the derivation of the filter for updating the decay rate constant a.

$$\frac{\partial h(n;a,\omega)}{\partial a} = n\, a^{n-1} \cos\omega n$$

$$= \frac{n}{a}\, a^n \cos\omega n \qquad (3.46)$$

$$= Z^{-1}\left(-\frac{z}{a}\frac{d}{dz}\left\{ \frac{z\,(z - a\cos\omega)}{z^2 - 2z\,a\cos\omega + a^2} \right\} \right)$$

Computing the derivative

$$\frac{V_a(z)}{X(z)} = \frac{z\,a^2\cos\omega - 2z^2\,a + z^3\cos\omega}{(\,z^2 - 2z\,a\cos\omega + a^2\,)^2} \qquad (3.47)$$

$$= \frac{z^{-1}\cos\omega - 2z^{-2}\,a + z^{-3}\,a^2\cos\omega}{1 - 4z^{-1}\,a\cos\omega + z^{-2}\,(4a^2\cos^2\omega + 2a^2) - 4z^{-3}\,a^3\cos\omega + z^{-4}\,a^4} \qquad (3.48)$$

where $X(z) = f[net(z)]$ as defined previously. Corresponding difference equation to obtain $V_a(n)$ is

$$v_a(n) = 4a\, v_a(n-1)\cos\omega - a^2\, v_a(n-2)\,[4\cos^2\omega + 2] + 4a^3\, v_a(n-3)\cos\omega - a^4 v_a(n-4)$$

$$+ \quad f[net(n-1)]\cos\omega - 2a\, f[net(n-2)] + a^2 f[net(n-3)]\cos\omega \qquad (3.49)$$

2. The DERIVATIVE filter $\frac{V_\omega(z)}{X(z)}$

In the same manner as in section III.C.1. we evaluate the partial derivative of $h(n;a,\omega)$ with respect to ω.

$$\frac{\partial h(n;a,\omega)}{\partial \omega} = -n\, a^n \sin\omega n$$

$$= Z^{-1}\left(z \frac{d}{dz}\left\{ \frac{a\, z \sin\omega}{z^2 - 2a\, z \cos\omega + a^2} \right\} \right)$$

Computing the derivative we have

$$\frac{V_\omega(z)}{X(z)} = \frac{\sin\omega\,(z\, a^3 - z^3\, a)}{(z^2 - 2z\, a \cos\omega + a^2)^2} \qquad (3.50)$$

$$= \frac{\sin\omega\,(z^{-3}\, a^3 - z^{-1}\, a)}{1 - 4z^{-1}\, a \cos\omega + z^{-2}\, a^2\,(4\cos^2\omega + 2) - 4z^{-3}\, a^3 \cos\omega + z^{-4}\, a^4} \qquad (3.51)$$

Hence, we implement the filter using the following difference equation

$$v_\omega(n) = 4a\, v_\omega(n-1)\cos\omega - a^2\, v_\omega(n-2)\,[4\cos^2\omega + 2] - 4a^3\, v_\omega(n-3)\cos\omega$$
$$+ a^4\, v_\omega(n-4) + \sin\omega\,(\, a^3 f[net(n-3)] - a\, f[net(n-1)] \qquad (3.52)$$

The structure of the two filters is shown in fig. 3.6a and 3.6b.

3. Conditions for the stability of the DERIVATIVE filters

The stability of the two filters is critically dependent on the parameter a. Since both the filters have the same denominator we need to perform the Jury's test only once in order to determine the constraints on a. In order that the filter be stable the first condition in Jury's test requires that

(i) $D(1) = 1 - 4a\cos\omega + a^2\,(4\cos^2\omega + 2) - 4a^3\cos\omega + a^4 > 0 \qquad (3.53.1)$

for all ω, since we place constraints only on a.

(ii) The second condition calls for making

$$(-1)^4 D(-1) = 1 + 4a\cos\omega + a^2\,(4\cos^2\omega + 2) + 4a^3\cos\omega + a^4 > 0 \qquad (3.53.2)$$

Next, we write down Jury's table for the other conditions

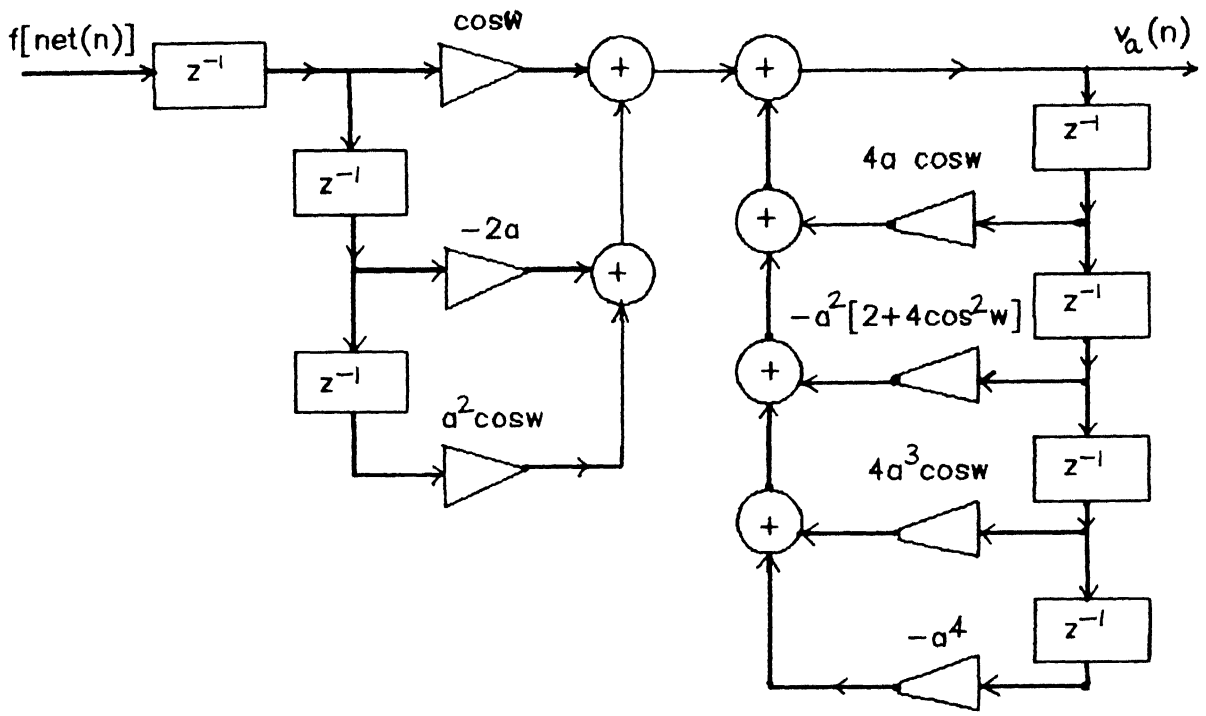


Figure 3.6(a) The DERIVATIVE filter to obtain $v_{a(n)}$ in case (c). [See eqn. 3.49]

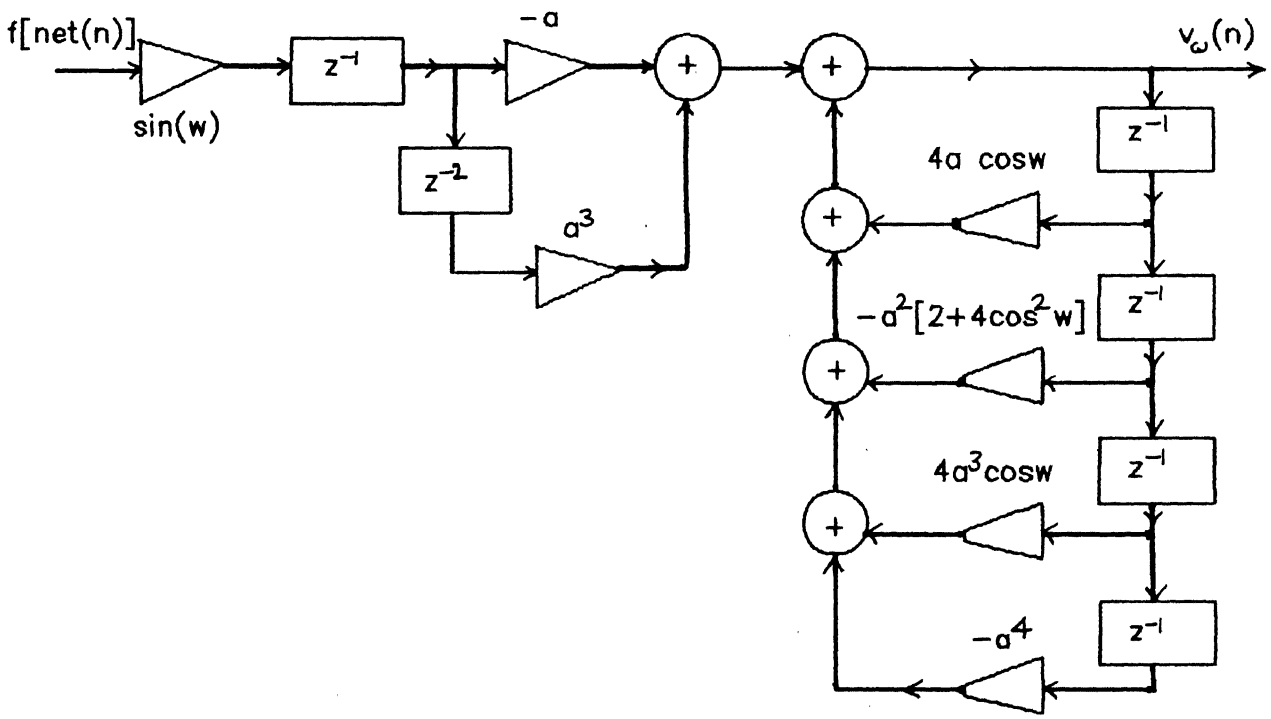


Figure 3.6(b) The DERIVATIVE filter to obtain $v_{\omega(n)}$ in case (c). [See eqn. 3.52]

Table 3.1

Jury's Table

ROW	COEFFICIENTS				
1	1	$-4a\cos\omega$	$4a^2\cos^2\omega + 2a^2$	$-4a^3\cos\omega$	a^4
	a^4	$-4a^3\cos\omega$	$4a^2\cos^2\omega + 2a^2$	$-4a\cos\omega$	1
2	$(1-a^8)$	$4a\cos\omega(a^6-1)$	$2a^2(1-a^4)(2\cos^2\omega + 1)$	$4a^3\cos\omega(a^2-1)$	
	$4a^3\cos\omega(a^2-1)$	$2a^2(1-a^4)(2\cos^2\omega + 1)$	$4a\cos\omega(a^6-1)$	$(1-a^8)$	
3	$(1-a^8)^2 - 16a^6\cos\omega\,(1-a^2)^2$	$8a^5\cos\omega(1-a^2)(1-a^4)(2\cos^2\omega + 1)$ $- 4(1-a^8)(1-a^6)a\cos\omega$	$2a^2(1-a^8)(1-a^4)(2\cos^2\omega+1)$ $- 16a^4(1-a^6)(1-a^2)\cos\omega$		

From the table the other constraints on a are

(iii) $1 > |a^4|$ (3.53.3)

(iv) $|1-a^8| > |4a^3\cos\omega(a^2-1)|$ (3.53.4)

(v) $|(1-a^2) - 16a^6\cos\omega(1-a^2)2| >$

$$2a^2|(1-a^8)(1-a^4)(2\cos^2\omega+1) - 8a^2(1-a^6)(1-a^2)\cos^2\omega| \qquad (3.53.5)$$

Conditions (i), (ii), (iii) directly yield $|a| < 1$. Looking at (iv) for $a > 0$ we have

$$1-a^8 > 4a^3(1-a^2)$$

or $$\frac{1}{4} > \frac{a^3(1-a^2)}{1-a^8}$$

which yields $a <= 1$. For $a < 0$

$$1-a^8 > 4a^3(1-a^2)$$

which once again yields $a >= -1$.

Finally, on looking at (v) we can write the following stronger inequality

$$|(1-a^8)2 - 16a^6(1-a^2)2| > 2a^2|(1-a^8)(1-a^4) - 8a^2(1-a^6)(1-a^2)| \qquad (3.54)$$

which finally yields $|a| < 0.6$ approximately on solving numerically. This is probably a conservative estimate. However, from our simulation we found that it is very close to the exact limit on a. Since, in practice, our signals are finite duration

and bounded, the filter for $v_a(n)$ and $v_w(n)$ never become unstable. However, if

$$|a| \gtrsim 0.6$$

then for large enough time instants the values of $v_a(n)$ and $v_w(n)$ become very large, hence creating finite precision and other computational problems. Keeping this in mind we limited the value of $|a|$ to a maximum of 0.7 and most of the time to less than 0.6.

We present the results of our simulation in the next chapter.

Chapter 4
Simulation Results

I. Introduction

ALFANS was designed with the aim of performing spatio-temporal pattern recognition. In order to test how well this network could perform the said task, a number of computer simulation experiments were designed and carried out. Each experiment was designed to see if the network could fulfil some elementary objectives which collectively define some aspects of spatio-temporal pattern recognition. We present these experiments, first defining the objectives and then describing the results. The experiments are discussed in the increasing order of their complexity.

We also present the results of some experiments we performed to look into certain other aspects of network behaviour and characteristics of the learning algorithm, like convergence, stability, speed of learning, and the kinds of patterns the network is capable (or incapable) of recognising.

II. Learning experiments with single patterns

A. Objectives

The main objectives are listed below.

(a) To examine whether the network has the capacity to learn a single pattern using the rule derived in chapter 3. By learning we mean, here, the ability of the algorithm to adjust the network weights to minimise the error E **"sufficiently."** The term "sufficiently" can be quantified using an acceptability criterion. In our experiments we assumed that a reduction in error E by 50% per pattern was

"sufficient" to claim that the network had learnt the input-target map or the input pattern. The value of 50% is on the higher side and the choice was motivated after noting the reduction in error E in a large number of recognition experiments involving many pairs of input-target patterns.

(b) The second objective was to test if the network could acquire the ability to perform non-linear prediction of the target pattern "sufficiently" well. Once again we use an acceptability criterion to say that the prediction error based on equation 3.5 must be less than about 10% of the value at start for that pattern. The value 10% is based on common experience and can be linked to the signal to prediction noise ratio as well as the rate-distortion function in some way.

(c) It is also important to examine whether the learning algorithm is stable and convergent, ie. whether the weights and parameters of the network stabilise, and the conditions under which this may be achieved.

(d) To examine what kind of patterns a network of this sort can learn.

B. Observations and Discussions

In temporal pattern recognition experiments we trained a 4 layer network with 1 neuron in the 4th or output layer, 5 in the 3rd and 2 in the second layer. The first layer being the input layer had no neurons [see fig. 3.2]. There was one input node. The rationale for this choice lay in the amount of computer time we could spend in getting a reasonably successful test performance. Too large a network means too much time spent on a single learning cycle while too small a network could lead to loss in performance or inhibit learning. The layer previous to the output layer was chosen to have a large enough number of neurons to improve the prediction performance of the network since the output unit performs

some kind of a synthesis using the signals from the previous layer which has five neurons in our simulations.

The activation function for the inner layers was chosen as

$$f_{(x)} = \frac{1}{1 + \exp(-g x)} \qquad (4.1)$$

where g is a constant that defines the hardness of the non-linearity which after trial and error was given a value 0.7 for satisfactory performance on recognition experiments without the net going into saturation every now so often.

For the outer layer, however, the activation function was chosen as

$$f_{(x)} = \frac{l}{1 + \exp(-g x)} \qquad (4.2)$$

where l is a constant and defines the level of the output. This was incorporated in order to train patterns which were not normalized. Level was chosen as 4.00 after trial and error.

In the experiments described below simulations were carried out using linear second order filters with decaying co-sinusoidal impulse responses corresponding to case (c) in section 3.IV.C.

We now present the results of 6 experiments chosen out of the many that were performed. Tables 4.2 (a) through 4.2 (e) list the learning rates η, η_ω and η_a used in the simulation programs.

layer	Neurons	η	η_ω	η_a
4	1	0.1	0.1	0.1
3	5	0.1	0.1	0.1
2	2	0.1	0.1	0.1
1	1	-	-	-

Table 4.2 (a) Learning rates used in experiments of sections II and III.

layer	Neurons	η	η_ω	η_a
4	1	0.08	0.03	0.03
3	5	0.1	0.05	0.05
2	2	0.2	0.05	0.05
1	1	-	-	-

Table 4.2 (b) Learning rates used in experiments of sections II and III.

layer	Neurons	η	η_ω	η_a
4	2	0.02	0.01	0.001
3	5	0.05	0.02	0.005
2	2	0.1	0.02	0.008
1	2	-	-	-

Table 4.2 (c) Learning rates used in experiments of section II and III.

layer	Neurons	η	η_ω	η_a
4	1	0.0001	0.0002	0.000008
3	5	0.00005	0.0002	0.000005
2	2	0.00002	0.0001	0.000001
1	1	-	-	-

Table 4.2 (d) Learning rates used in experiments of section II and III.

layer	Neurons	η	η_ω	η_a
4	1	10^{-7}	10^{-6}	8×10^{-7}
3	5	5×10^{-7}	2×10^{-6}	5×10^{-7}
2	2	10^{-6}	2×10^{-6}	2×10^{-7}
1	1	-	-	-

Table 4.2 (e) Learning rates used in experiments of section II and III.

1. Experiment 1

This was the first experiment we performed with temporal patterns. For simplicity we chose sinusoids or their linear combinations as the input and target patterns.

Input : insine1 : 0.8sin(0.4n)

Target : outsine1 : 0.8sin(0.4n) + 0.4sin(0.8n) + 0.24sin(1.2n)

Test inputs :

[1] ibsine1 : 0.8sin(0.4n) + Unoise(n)

[2] insine2 : 0.8sin(0.6n)

[3] insine3 : 0.8sin(0.5n)

where Unoise(n) is zero mean uniform noise of variance $\frac{1}{12}$ over [-0.5, +0.5]. These patterns are shown in figures 4.1(a) through (c).

Error E at start = 44.65

Error reduced to 10.332 on the 96 learning cycle.

Thereafter, the error increased drastically and fluctuated randomly between 438 and 1466. However, when the learning rates were reduced considerably to those shown in table 4.2(e) the weights and parameters (see table 4.2.1) showed little change and the error E decreased steadily, empirically suggesting that the algorithm has convergent property for low enough learning rates although it is divergent for large values of η, η_ω and η_a. This is quite in accordance with the fact that in the derivation of the algorithm the increments in weights and parameters were assumed to be infinitesimal changes because they were made proportional to the derivative of the error E. What remains unknown are the bounds on the learning rate constants and other conditions which can guarantee convergence.

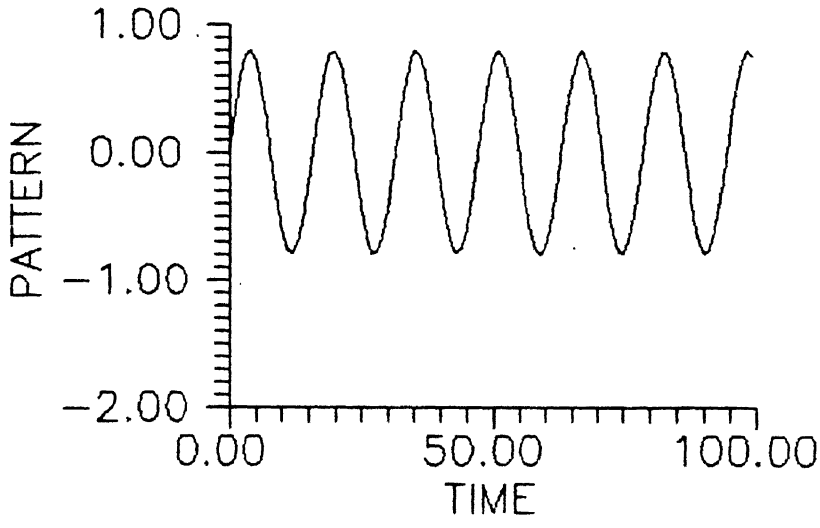


Figure 4.1 (a) Input : insine1 : 0.8sin(0.4n)

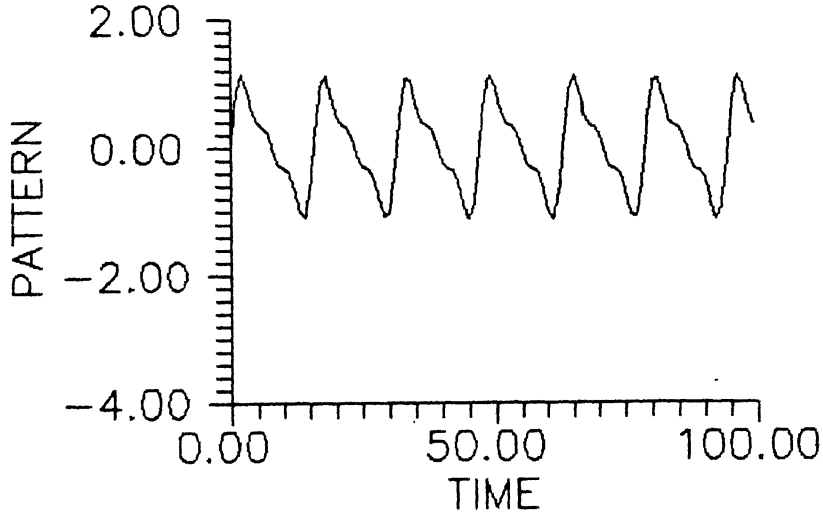


Figure 4.1 (b) Target : outsine1 : 0.8sin(0.4n) + 0.4sin(0.8n) + 0.24sin(1.2n)

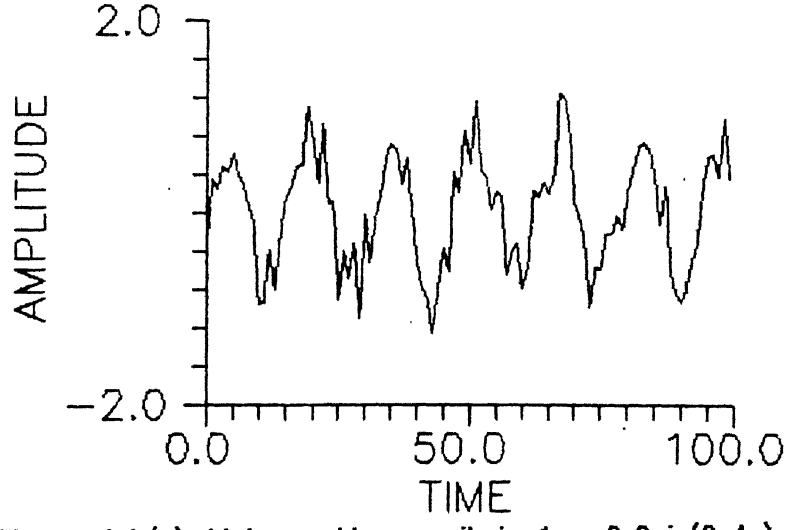


Figure 4.1 (c) Noisy pattern : ibsine1 : 0.8sin(0.4n) + Unoise(n)

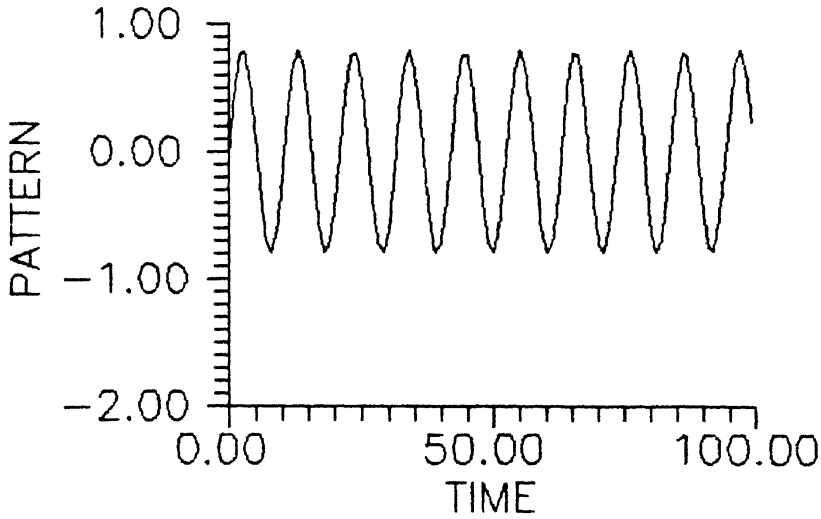


Figure 4.2(a) Input : insine2 : 0.8sin(0.6n)

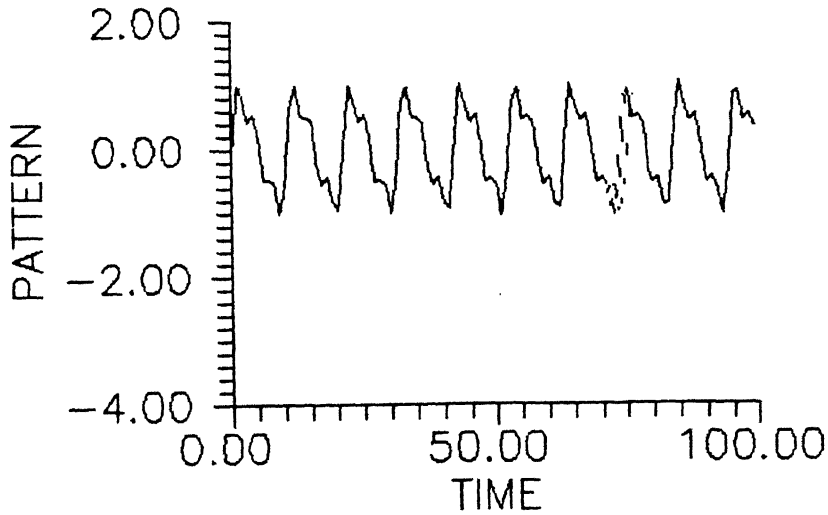


Figure 4.2(b) Target : outsine2 : 0.8sin(0.6n) + 0.24sin(1.2n) + 0.32sin(1.8n)

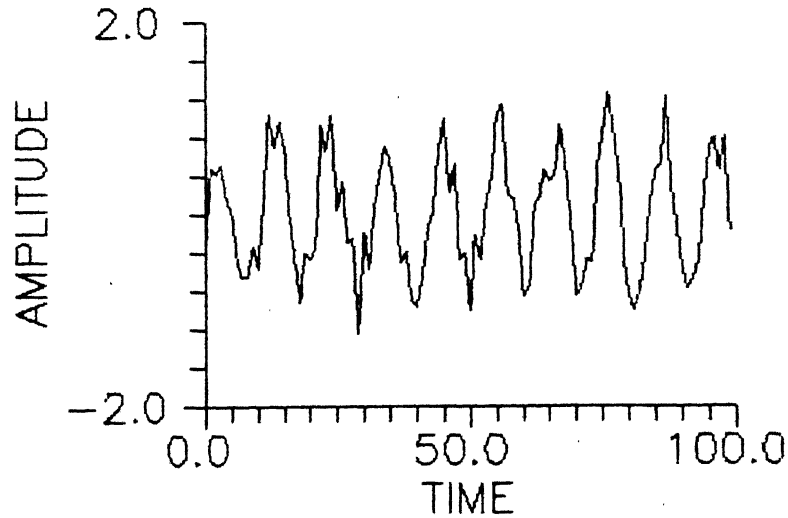


Figure 4.2(c) Noisy pattern : ibsine2 : 0.8sin(0.6n) + Unoise(n)

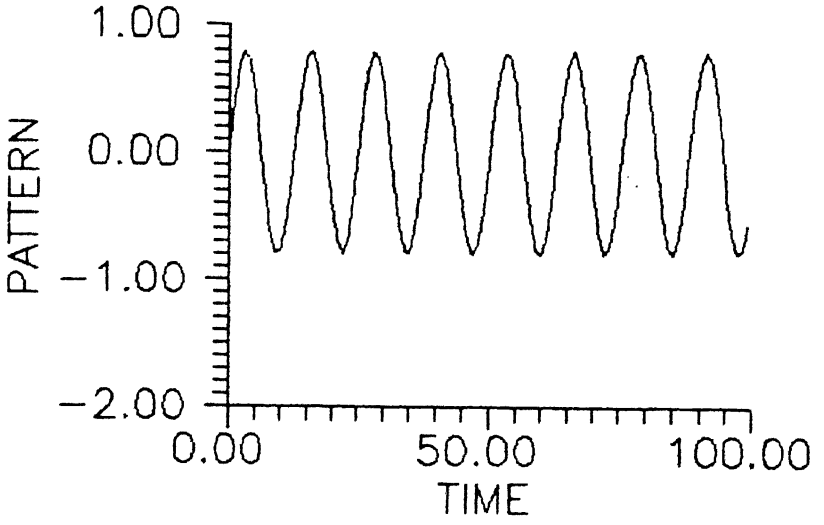


Figure 4.3(a) Input : insine3 : 0.8sin(0.5n)

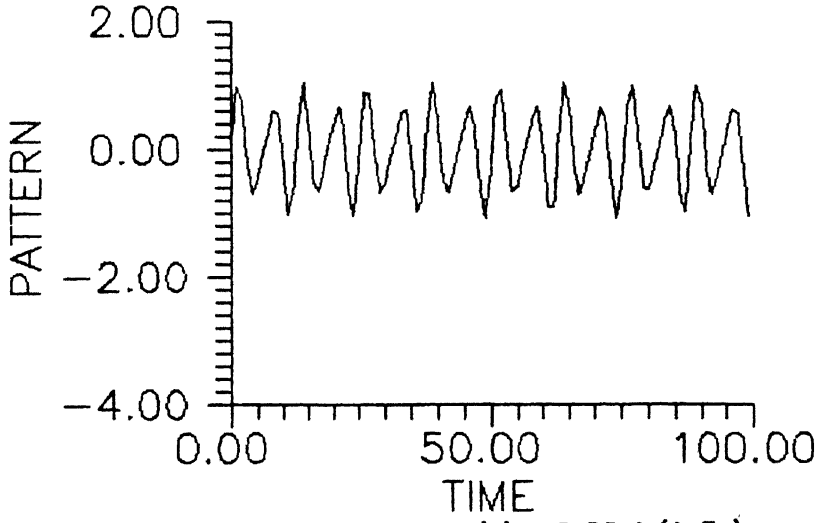


Figure 4.3(b) Target : outsine3 : 0.8sin(n) + 0.32sin(1.5n)

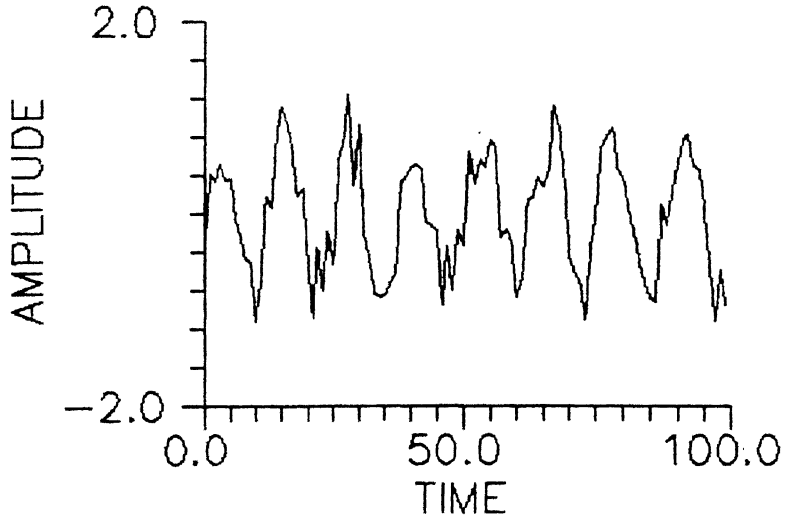


Figure 4.3(c) Noisy pattern : ibsine3 : 0.8sin(0.5n) + Unoise(n)

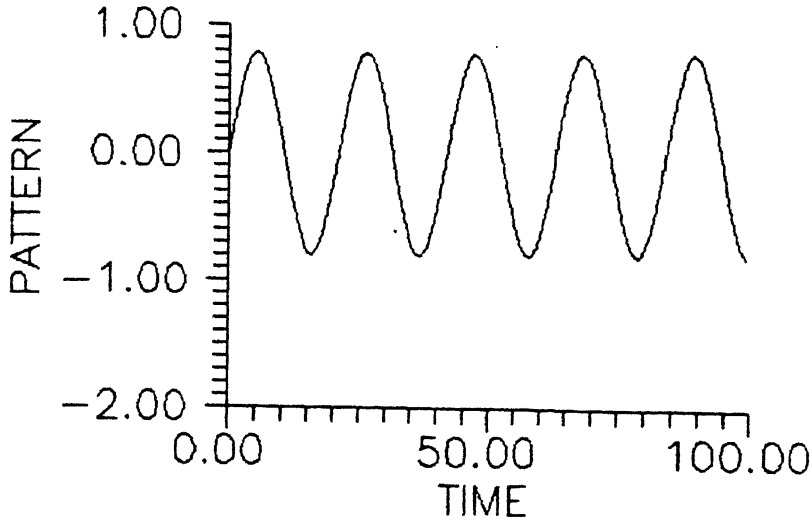


Figure 4.4(a) Input : insine4 : 0.8sin(0.3n)

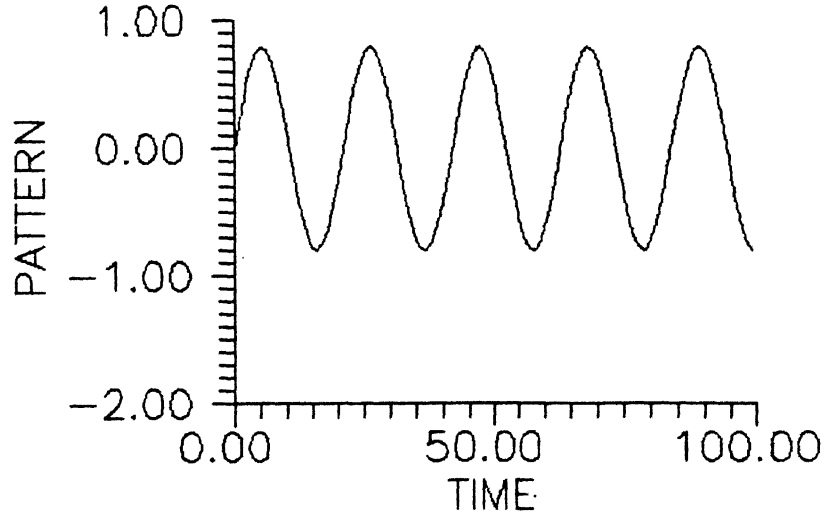


Figure 4.4(b) Target : outsine4 : 0.8sin(0.3n)

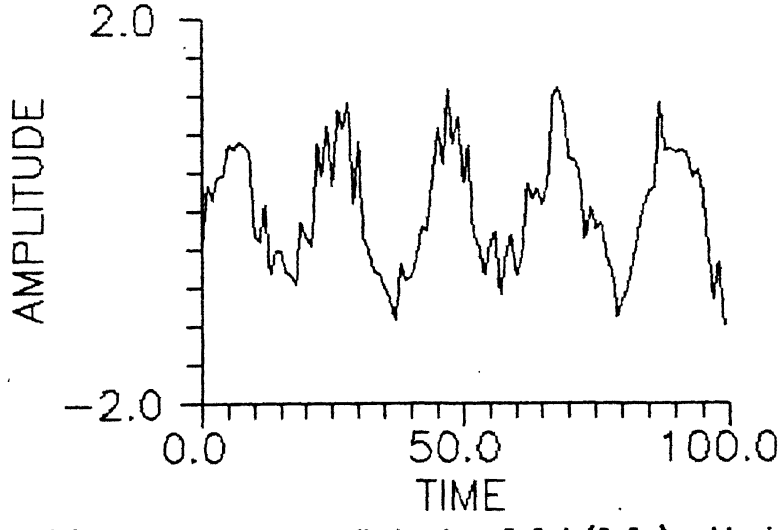


Figure 4.4(c) Noisy pattern : ibsine4 : 0.8sin(0.3n) + Unoise(n)

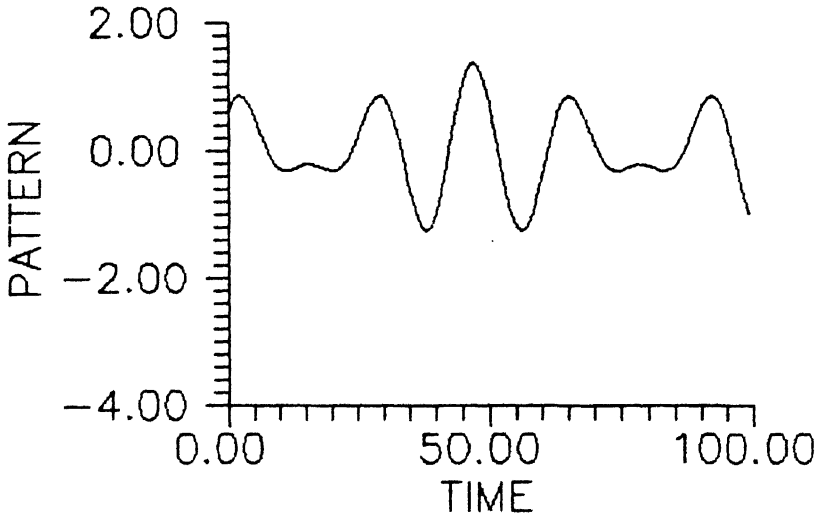


Figure 4.5(a) Input : insine5 : 0.8sin(0.3n) + 0.6cos(0.4n)

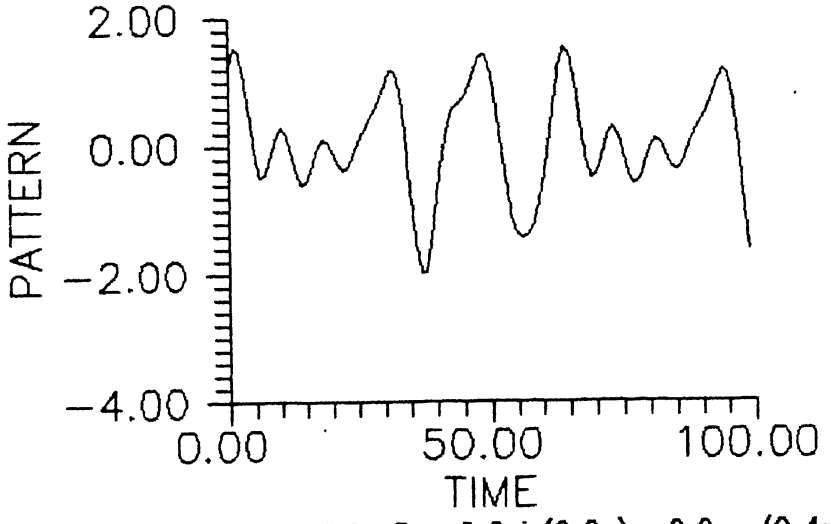


Figure 4.5(b) Target : outsine5 : 0.8sin(0.3n) + 0.8cos(0.4n) + 0.4cos(0.6n)
+ 0.32sin(0.8n)

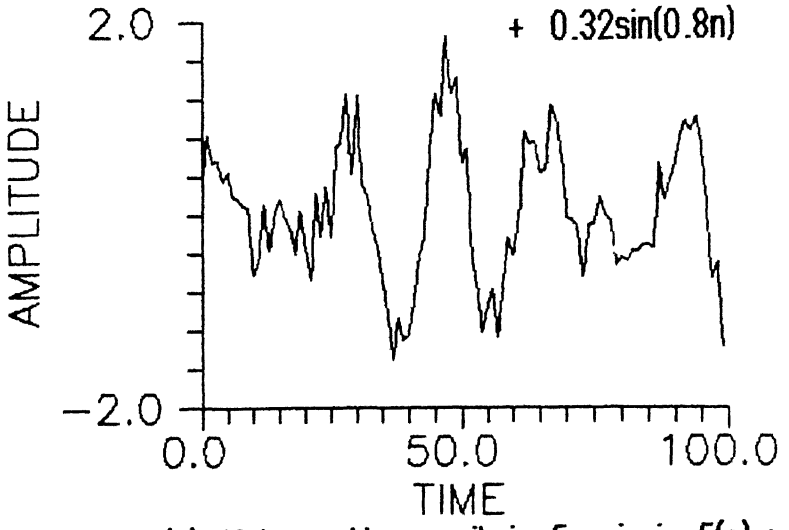


Figure 4.5(c) Noisy pattern : ibsine5 : insine5(n) + Unoise(n)

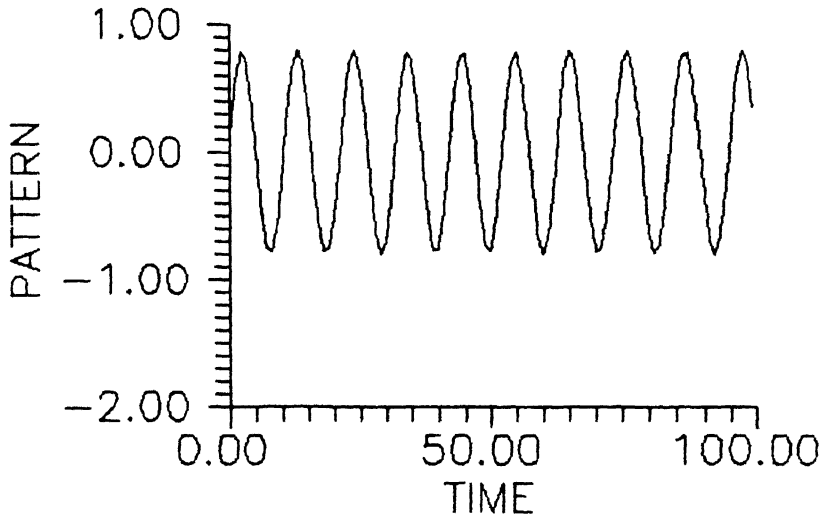


Figure 4.6(a) Input : insine6 : 0.8sin[0.6n + 0.1cos(0.1n)]

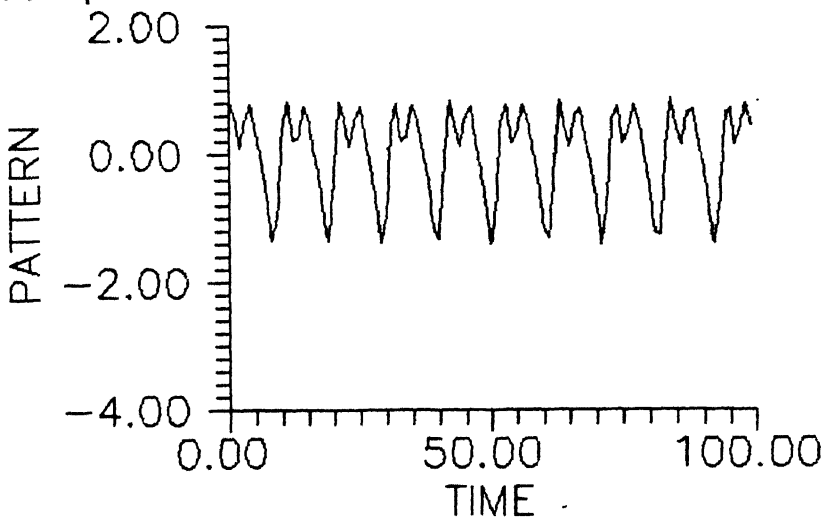


Figure 4.6(b) Target : outsine6 : 0.8sin(0.6n) + 0.5cos(1.2n) + 0.3cos(1.8n)

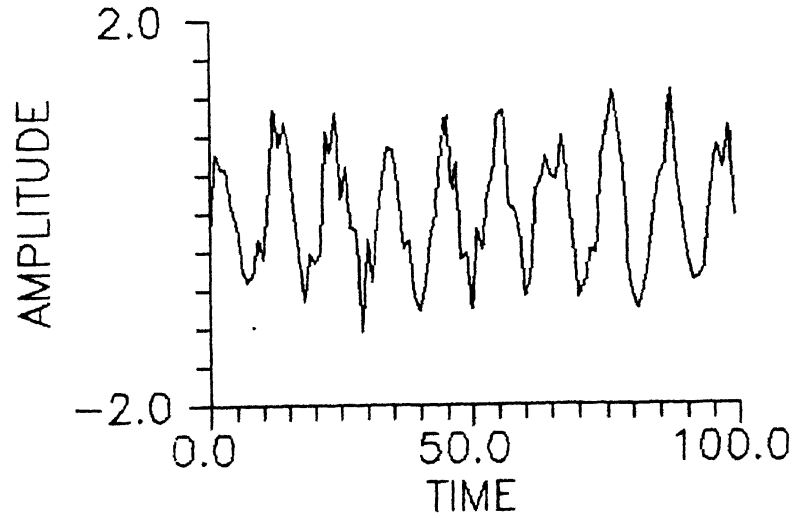


Figure 4.6(c) Noisy pattern : ibsine6 : 0.8sin[0.6n + 0.1cos(0.1n)] + Unoise(n)

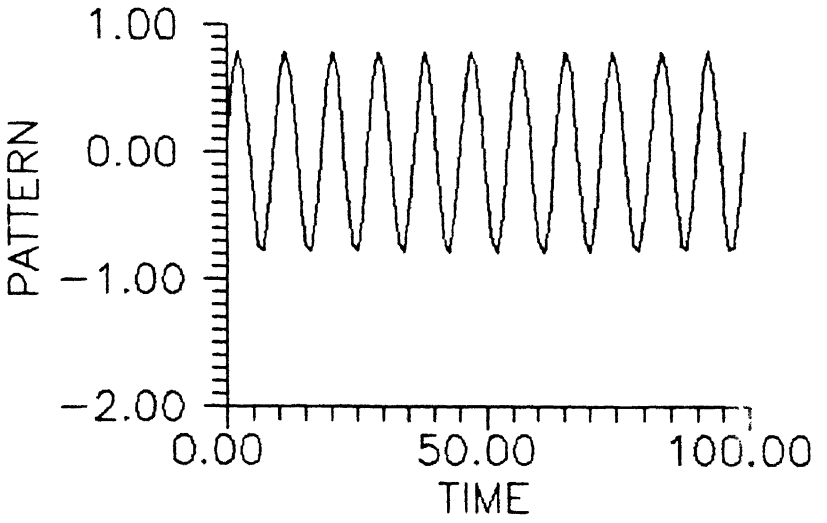


Figure 4.7(a) Input : insine7 : 0.8cos[0.7n + 0.1cos(0.1n)]

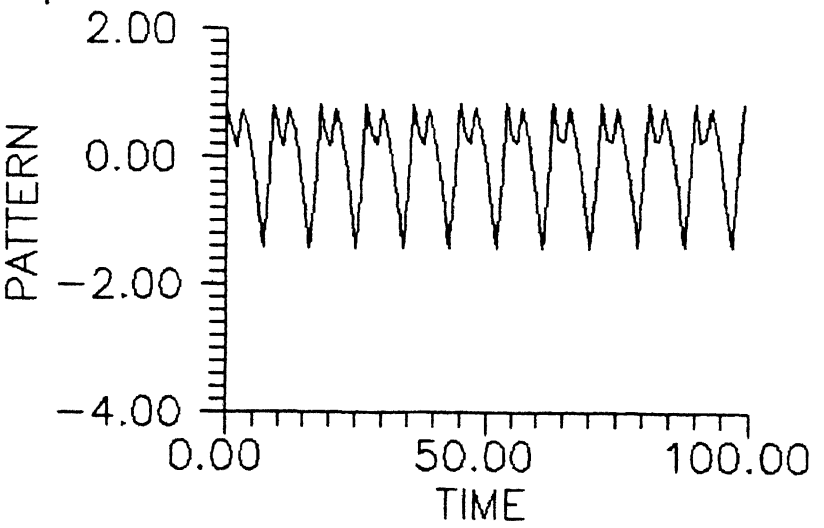


Figure 4.7(b) Target : outsine7 : 0.8sin(0.7n) + 0.5cos(1.4n) + 0.3cos(2.1n)

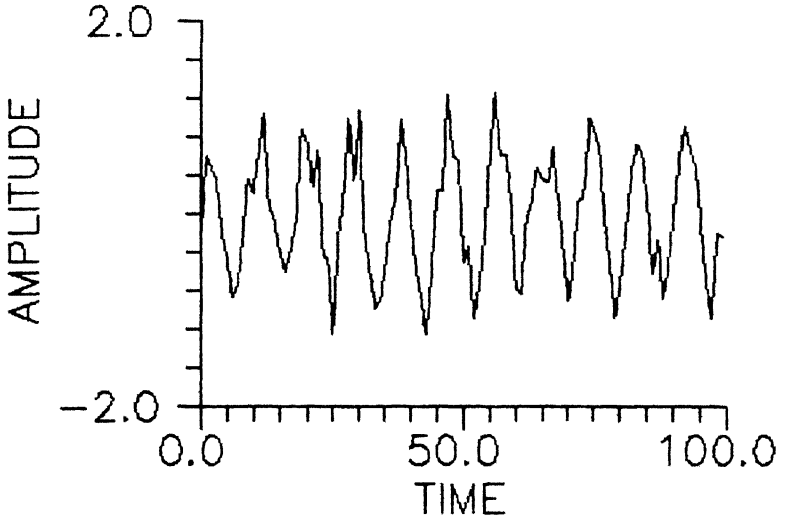


Figure 4.7(c) Noisy pattern : ibsine7 : 0.8cos[0.7n + 0.1cos(0.1n)] + Unoise(n)

Error = 10.332334

	I1	I2	J1	J2	J3	J4	J5	K1
ω	0.79151	-5.81689	3.71686	2.12528	1.48737	0.23398	0.81328	0.60261
a	-0.50000	0.50000	-0.50000	-0.17758	0.02157	0.49924	0.24089	-0.50000

WEIGHTS

w_{ji}	1.96172			1.37314				
	0.75392	0.93801		0.24801	0.08726		0.25352	0.11491
	0.23913	0.17806		0.23720	0.12022			
w_{kj}	18.97129		0.07855	0.09317		0.08700	0.08666	

Table 4.2.1 Weights and filter parameters of experiment 1 (section II)

Further, we may note that the prediction error is too large and can conclude that for this choice of linear filter and learning rates the network is not a good predictor in this case. However, the network fulfils the first objective. This can be easily verified by feeding a noise corrupted version of insine1. We used zero mean uniform noise of variance $\frac{1}{12}$ in the interval [-0.5,0.5] to corrupt insine1. This corresponds to a signal to noise ratio (SNR) of roughly 5.8 dB which is fairly low. Yet, upon testing with the corrupted pattern ibsine 1, the network produced an error E = 12.148 which is very close to the value that was obtained using insine1 and is still less than $\frac{1}{3}$ of the error at start, i.e. 44.65.

The same network, on feeding another input pattern insine3 yielded a very large error E = 64.55 and on input pattern insine2 it yielded E = 74.30 which is again very high.

This suggest that the network could be trained to distinguish a class of patterns from all others. This can be also seen in the experiments that follow.

2. Experiment 2

In this experiment we trained the network to recognise a narrow band FM wave. The target pattern was chosen to have a component with the same frequency as that of the carrier of the input FM wave. The weights and filter parameters obtained on training are shown in table 4.2.2.

Input : insine6 : 0.8sin[0.6n + 0.1cos(0.1n)] : A narrow band FM signal.

target : outsine6 : 0.8sin(0.6n) + 0.5cos(1.2n) + 0.3cos(1.8n)

Test inputs :

[1] ibsine6 : 0.8sin[0.6n + 0.1cos(0.1n)] + Unoise(n) : Noisy input which is centered around the carrier of the input FM wave.

[2] insine2 : 0.8sin(0.6n)

Error at start = 50.259

Error reduced to 17.55

It can be seen once again that although the target pattern prediction is poor the extent of learning is quite satisfactory. On feeding the noise corrupted version of insine6 the error produced is E = 22.012 which is close to what was produced using the training pattern insine6.

What is most interesting is that on testing with pattern insine2 the network yielded an error E = 17.204 which is less than the value obtained using the original FM wave insine6. A convincing explanation for this phenomenon is that the unmodulated tone insine2 is closer to the actual exemplar of the class the network learned to distinguish. The FM wave can be considered to have been treated as a

Error = 17.554953

	I1	I2	J1	J2	J3	J4	J5	K1
ω	1.60777	1.09986	1.54627	2.11377	1.48761	0.24973	0.82746	3.44014
a	0.50000	0.50000	-0.50000	-0.12355	0.00398	0.25264	0.14498	-0.50000

WEIGHTS

w_{il}	1.67141			1.05147				
w_{ji}	2.64912	0.65412		0.20430	0.07117		0.20551	0.09696
	0.17008	0.15486		0.18639	0.10179			
w_{kj}	7.86779		0.07855		0.09317	0.08700	0.08666	

Table 4.2.2 Weights and filter parameters of experiment 2 (section II)

noisy version of the exemplar where a small random phase variation has been introduced. This illustsrates the fact that the network is a frequency selective network more or less and classifies patterns based on the input and target pattern frequencies. This fact will be further exemplified in some of the experiments that follow where the network refuses to learn maps between input and target which have no common frequency components or other characteristics.

3. Experiment 3

The results of this experiment illustrate the prediction capabilities of this network. The weights and parameters of the trained network are shown table 4.2.3.

Input : insine4 : 0.8sin(0.3n)

Target : outsine4 : 0 .8sin(0.3n)

Test input : ibsine4 : 0.8 sin(0.3n) + Unoise(n)

where U noise(n) is once again zero mean uniform noise in the interval [-0.5,0.5] with variance $\frac{1}{12}$.

Error at start = 32.038

On testing with noisy input ibsine4 error was E=0.2614 which is quite small compared to the starting value E = 32.038. This shows that if the algorithm is a converging one (we have no theoretical justification to claim its convergence or divergence characteristics) then for low enough learning rates the error E would steadily decrease to a value E_∞ as number of learning cycles become very large. It is then obvious that the system would predict at best with an error E_∞ and no less. The value of E_∞ in this case, is probably close to zero. Hence the network proved to be a very good predictor here and the error E, after training, is within 10% of its value at start, as stipulated in section II.A. More about prediction is discussed in some experiments that follow.

Error = 0.2614

	I1	I2	J1	J2	J3	J4	J5	K1
ω	0.79151	-5.81689	3.71686	2.12528	1.48737	0.23398	0.81328	0.60261
a	-0.50000	0.50000	-0.50000	-0.17758	0.02157	0.49924	0.24089	-0.50000

WEIGHTS

w_{il}	1.96172			1.37314				
w_{ji}	0.75392	0.93801		0.24807	0.08726		0.25355	0.11490
	0.23913	0.17806		0.23720	0.12022			
w_{kj}	18.97129		0.07855		0.09317	0.08700	0.08666	

Table 4.2.3 Weights and filter parameters of experiment 3 (section II)

4. Experiment 4

This experiment illustrates the fact that the network is a frequency selective one. So far we chose the input and target patterns with the restriction that there be at least one component sinusoid in the target that has the same frequency as the predominant component in the input. In this experiment we chose the target pattern to contain only the higher harmonics of the input sinusoid as described below.

Input : insine3 : 0.8sin(0.5n)

Target : outsine3 : 0.8sin(1.0n) + 0.32sin(1.5n)

Error at start = 37.090099

Error reduced to 37.090061 in 1446 iterations on using the learning rates shown in table 4.2(d). One can see that the network hardly learnt at all. We carried on learning for another 378 iterations using the rates shown in table 4.2(e) but the error did not reduce at all.

This suggests that the network cannot learn maps between patterns with widely differing characteristics. The same conclusion can be drawn from the results of experiment 6 to be discussed later.

5. Experiment 5

In this experiment we shifted to another class of input and target patterns. Here, we used decaying exponentials to train the network and it was quite a success. The weights and filter parameters are shown in table 4.2.4.

Input : exp2 : 0.8exp(-0.1n)

Target : expout2 : 1.2exp(-0.2) + 0.6exp(-0.4n)

Error at start = 0.425888

Error reduced to 0.092013 after 752 iterations.

This shows that the network is not only capable of learning sinusoidal patterns but can also learn patterns like decaying exponentials provided the input and target have common characteristics.

Error = 0.092013

	I1	I2	J1	J2	J3	J4	J5	K1
ω	3.82364	0.68382	8.50577	2.11672	1.48763	0.24490	0.82421	2.35365
a	0.50000	-0.50000	-0.50000	-0.13502	0.01379	0.36272	0.17379	-0.50000

WEIGHTS

w_{il}	0.55522			1.13138				
w_{ji}	4.02528	0.76624		0.21471	0.06320		0.21569	0.08727
	0.19234	0.14758		0.19915	0.09316			
w_{kj}	16.79617		0.07855	0.09317		0.08700	0.08666	

Table 4.2.4 Weights and filter parameters of experiment 5 (section II)

6. Experiment 6

In this experiment we chose the input and target from two different classes to illustrate that learning is unlikely to occur in the first place. The input and target hardly have any common characteristics here.

Input : exp1 : 0.8exp(-0.2n)

Target : expout1 : 0.8sin(0.6n) + 0.4cos(1.2n) + 0.02sin(3n)

Error at start = 47.678257

The error remained at this value even after 213 iterations when training was carried out using the learning rates shown in table 4.2(d).

Thus, it seems empirically that the only criterion for learning to be successful in case of ALFANS is that both input and output patterns must have some common characteristics. Incase of mixtures of sinusoids or square waves as inputs, the targets must have at least one of its components matching in frequency with one of the components of the input, as mentioned earlier. Experiments with square waves were also carried out and some of them have been discussed in section III which deals with pattern recognition. Our experiments also suggest that target patterns must be of a low pass nature with not too many higher harmonics. This is because prediction of abrupt changes in pattern with time is very difficult due to the low pass nature of the network.

Since single pattern learning experiments were by and large successful and yielded much insight into the network behaviour, learning process, and the suitability of certain patterns as inputs and targets, we proceeded to carry out pattern recognition experiments in which the network was trained to recognise and classify the input patterns. Proper prediction of the desired target could not be guaranteed as we discovered in the experiments described so far.

III Temporal pattern recognition experiments

In these experiments the objectives (a) through (d) listed in section II.A remain as important. Apart from these there is one singularly most important objective, namely to see whether the network can recognise an input pattern to be belonging to one out of a number of class that it has been trained for. By and large we found that if the pattern pairs could be taught to the network

individually as in section II the network could also perform recognition on these pattern pairs fairly well.

We shall first discuss temporal pattern recognition with single input-single output ALFANS. Then we shall discuss spatio-temporal pattern recognition in which the network has more than 1 node at both the input and the output.

1. Experiment 1

In this experiment we chose just two pattern pairs both of which could be taught to the network, individually.

Inputs :

[1] insine1 : 0.8sin(0.4n)

[2] insine2 : 0.8sin(0.6n)

Targets :

[1] outsine1 : 0.8sin(0.4n) + 0.4sin(0.8n) + 0.24sin(1.2n)

[2] outsine2 : 0.8sin(0.6n) + 0.24sin(1.2n) + 0.32sin(1.8n)

Test inputs :

[1] ibsine1 : 0.8sin(0.4n) + Unoise(n)

[2] ibsine2 : 0.8sin(0.6n) + Unoise(n)

[3] insine6 : 0.8sin[0.6 + 0.1cos(0.1n)]

On training with insine1 : outsine1 and insine2 : outsine2 using the learning rates in table 4.2(b),

Error at start = 84.605

Error reduced to 17.463 in the 114th learning cycle.

Thereafter, the learning rates were reduced to those shown in table 4.2(c). Error further reduced to 14.246 and stabilised around that value for very low learning

rates shown in table 4.2(e). The learned weights and filter parameters are shown in table 4.3.1 while figures 4.1(a)-(c), 4.2(a)-(c) and 4.3(a)-(c) show these patterns.

Then, we performed recognition tests usings the trained network. The results of the recognition experiment are tabulated in table 4.3.2. The first column shows the input fed to the network, the second column holds the target with which the output is compared, the third column shows the error E between the target and generated output. The target for which the error E is minimum, given the input, is the class to which that input belongs.

It is clearly seen that on feeding insine1 the error E between the generated output and the target outsine1 is much less than the error when outsine2 is specified as the target. This is again the case on feeding ibsine1 which is same as insine1 corrupted with additive uniform noise of mean zero and variance $\frac{1}{12}$ in the interval [-0.5,0.5], except that the error increased from 9.17 to 15.01, which is still quite small when compared to 72.74 which is the error E between output and outsine2 with ibsine1 at the input.

Error = 14.235181

	I1	I2	J1	J2	J3	J4	J5	K1
ω	1.79858	0.28189	91.3500	2.11522	1.49123	0.24862	0.82828	2.90972
a	-0.07537	-0.5000	-0.17635	-0.05913	-0.00642	0.11017	0.06117	0.46169

WEIGHTS

w_{jl}	4.75082			1.27432				
w_{ji}	-1.57750	0.45769		0.45962	0.20140		0.50480	0.24955
	0.45752	0.30029		0.46644	0.24458			
w_{kj}	-13.80250		0.07855	0.09317		0.08700	0.08666	

Table 4.3.1 Weights and filter parameters of experiment 1 (section III)

S.no.	Input	Target	Error	Class
1.	insine1	outsine1	9.17	outsine1
		outsine2	70.59	
2.	insine2	outsine1	75.42	
		outsine2	5.07	outsine2
3.	ibsine1	outsine1	15.01	outsine1
		outsine2	72.74	
4.	ibsine2	outsine1	76.56	
		outsine2	9.86	outsine2
5.	insine6	outsine1	76.79	
		outsine2	6.16	outsine2

Table 4.3.2 Results of experiment 1 (section III)

The pattern insine6 belongs to the second class as shown in table 4.3.2. This is once again as expected because insine6 is an FM wave with carrier frequency $\omega = 0.6$ which is also the frequency of the single tone insine2 for which outsine2 is the desired target. Thus, insine6 can be treated as insine2 with a small random term added to the phase.

2. Experiment 2

In this experiment we chose 3 pairs of patterns as listed below.

Inputs :

[1] insine1 : $0.8\sin(0.4n)$

[2] insine2 : $0.8\sin(0.6n)$

[3] insine4 : $0.8\sin(0.3n)$

Targets :

[1] outsine1 : $0.8\sin(0.4n) + 0.4\sin(0.8n) + 0.24\sin(1.2n)$

[2] outsine2 : $0.8\sin(0.6n) + 0.24\sin(1.2n) + 0.32\sin(1.8n)$

[3] outsine4 : 0.8sin(0.3n)

Test Inputs :

[1] ibsine1 : 0.8sin(0.4n) + Unoise(n)

[2] ibsine2 : 0.8sin(0.6n) + Unoise(n)

[3] ibsine4 : 0.8sin(0.3n) + Unoise(n)

[4] insine6 : 0.8sin[0.6n + 0.1cos(0.1n)]

[5] insqr1 : sgn[insine1(n)]

[6] insqr2 : sgn[insine2(n)]

[7] insqr4 : sgn[insine4(n)]

[8] insqr6 : sgn{0.8sin[0.6n + 0.5cos(0.1n)]}

Table 4.3.4 lists the results of the recognition experiment while table 4.3.3 holds the weights and filter parameters obtained on training the system. Input and target patterns are shown figures 4.1(a)-(c) through 4.7(a)-(c). In this experiment we also fed the four square waves listed above as test inputs and watched the excellent classification that the network performed.

Error = 29.387571

	I1	I2	J1	J2	J3	J4	J5	K1
ω	-2.76174	-1.55511	0.29410	2.12135	1.48814	0.24614	0.82035	-1.26787
a	-0.50000	0.50000	0.50000	-0.17854	0.02128	0.43928	0.23446	-0.50000

WEIGHTS

w_{jl}	-1.17759			0.84809				
	-0.99492	1.04034		0.28052	0.09615		0.29017	0.12512
	0.28564	0.18915		0.27426	0.13018			
w_{kj}	19.38893		0.07855		0.09317	0.08700	0.08666	

Table 4.3.3 Weights and filter parameters of experiment 2 (section III)

We can clearly see how the network is able to recognise the square waves to be belonging to the appropriate classes. Intuitively, a square wave derived from insine1 is closer to insine1 than to any other sinewave, especially since that square wave is composed of harmonics of only one single tone, namely insine1. The network distinguishes and classifies the input just on the basis of this fact.

S.no.	Input	Target	Error	Class
1.	insine1	outsine1	13.84	outsine1
		outsine2	79.00	
		outsine4	88.87	
2.	insine2	outsine1	78.78	
		outsine2	12.61	outsine2
		outsine4	81.75	
3.	insine4	outsine1	107.73	
		outsine2	96.28	
		outsine4	2.94	outsine4
4.	ibsine1	outsine1	14.15	outsine1
		outsine2	81.54	
		outsine4	97.87	
5.	ibsine2	outsine1	77.51	
		outsine2	15.22	outsine2
		outsine4	90.16	
6.	ibsine4	outsine1	97.99	
		outsine2	88.30	
		outsine4	5.88	outsine4
7.	insine6	outsine1	81.14	
		outsine2	14.09	outsine2
		outsine4	81.31	
8.	insqr1	outsine1	29.81	outsine1
		outsine2	122.52	
		outsine4	146.42	
9.	insqr2	outsine1	124.49	
		outsine2	27.88	outsine2
		outsine4	125.34	
10.	insqr4	outsine1	171.00	
		outsine2	154.49	
		outsine4	29.66	outsine4
11.	insqr2	outsine1	135.18	
		outsine2	43.91	outsine2
		outsine4	128.21	

Table 4.3.4 Results of experiment 2 (section III)

3. Experiment 3

This experiment was performed using 4 pairs of input and target patterns. Two of them were single tones and two were FM waves as listed below.

Inputs :

[1] insine1 : 0.8sin(0.4n)

[2] insine4 : 0.8sin(0.3n)

[3] insine6 : 0.8sin[0.6n + 0.1cos(0.1n)]

[4] insine7 : 0.8sin[0.7n + 0.1cos(0.1n)]

Targets :

[1] outsine1 : 0.8sin(0.4n) + 0.4sin(0.8n) + 0.24sin(1.2n)

[2] outsine4 : 0.8sin(0.3n)

[3] outsine6 : 0.8sin(0.6n) + 0.5cos(1.2n) + 0.3cos(1.8n)

[4] outsine7 : 0.8sin(0.7n) + 0.5cos(1.4n) + 0.3cos(2.1n)

Test inputs :

[1] ibsine1 : insine1(n) + Unoise(n)

[2] ibsine4 : insine4(n) + Unoise(n)

[3] ibsine6 : insine6(n) + Unoise(n)

[4] ibsine7 : insine7(n) + Unoise(n)

[5] insine2 : insine2(n) + Unoise(n)

Error at start = 175.21

Error reduced to 69.90

The results of the experiment are shown in table 4.3.5 while the weights are and filter parameters obtained on training are shown in table 4.3.6. The figures 4.8(a)-(d) show the generated outputs and how well they predict the target. As can be seen in these figures, the generated outputs are perfectly in phase with the

S.no.	Input	Target	Error	Class
1.	insine1	outsine1	16.72	outsine1
		outsine4	49.08	
		outsine6	56.58	
		outsine7	64.76	
2.	insine4	outsine1	64.32	
		outsine4	4.72	outsine4
		outsine6	64.73	
		outsine7	66.39	
3.	insine6	outsine1	51.67	
		outsine4	46.20	
		outsine6	23.46	outsine6
		outsine7	66.39	
4.	insine7	outsine1	60.95	
		outsine4	40.05	
		outsine6	61.77	
		outsine7	25.00	outsine7
5.	ibsine1	outsine1	15.00	outsine1
		outsine4	52.62	
		outsine6	57.13	
		outsine7	69.13	
6.	ibsine4	outsine1	59.99	
		outsine4	7.04	outsine4
		outsine6	63.01	
		outsine7	64.00	
7.	ibsine6	outsine1	49.59	
		outsine4	49.65	
		outsine6	24.27	outsine6
		outsine7	70.50	
8.	ibsine7	outsine1	56.94	
		outsine4	41.71	
		outsine6	60.14	
		outsine7	27.96	outsine7
9.	insine2	outsine1	50.71	
		outsine4	46.38	
		outsine6	22.95	outsine6
		outsine7	67.02	

Table 4.3.5 Results of experiment 3 (section III)

predominant component of the target. However, the prediction error E for each pattern was more than the acceptable limit of 10% as stipulated in section II.A. What is interesting is that the outputs generated by input patterns 1 through 4 do not at all match any targets other than the corresponding ones as can be seen in figures 4.8(a)-(d) and 4.9(a)-(d). Further, on feeding the noisy test patterns 1

through 4 shown in figures 4.1 through 4.7, the generated outputs are clean sinusoids with their phases matching the phases of their corresponding targets. This can be seen in figures 4.10(a)-(d).

Error = 69.903511

	I1	I2	J1	J2	J3	J4	J5	K1
ω	1.56039	1.65396	3.13122	2.11468	1.48714	0.24715	0.82598	4.91838
a	0.26609	0.08985	-0.50000	-0.11500	0.00575	0.26770	0.14011	-0.50000

WEIGHTS

w_{jI}	1.19647			0.86546				
	1.76212	0.57325		0.19333	0.06296		0.19156	0.08712
	0.16461	0.14679		0.17509	0.09290			
w_{kj}	8.76608		0.07855		0.09317		0.08700	0.08666

Table 4.3.6 Weights and filter parameters of experiment 3 (section III)

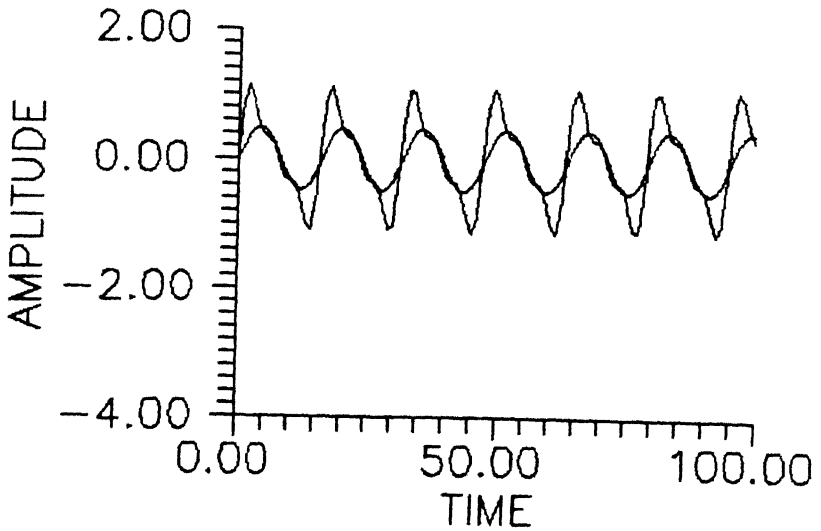


Figure 4.8(a) Target outsine1 and the output due to insine1

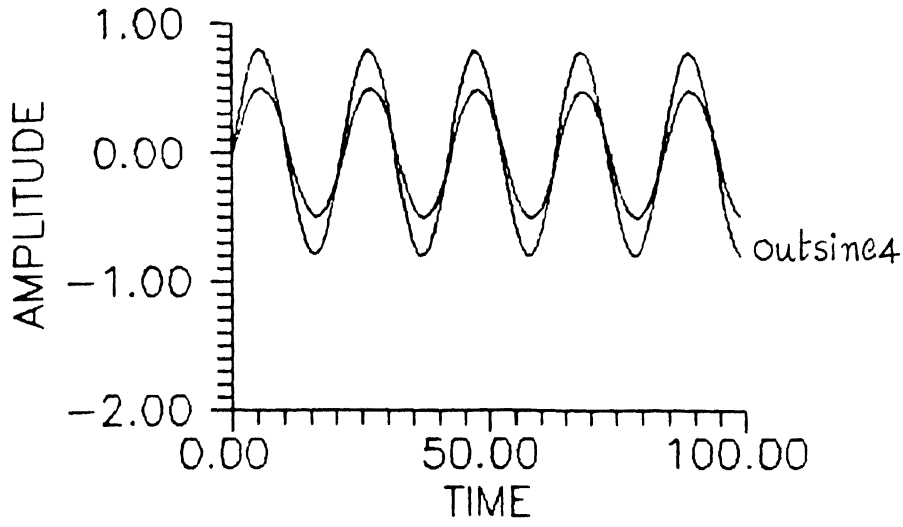


Figure 4.8(b) Target outsine4 and the output due to insine4

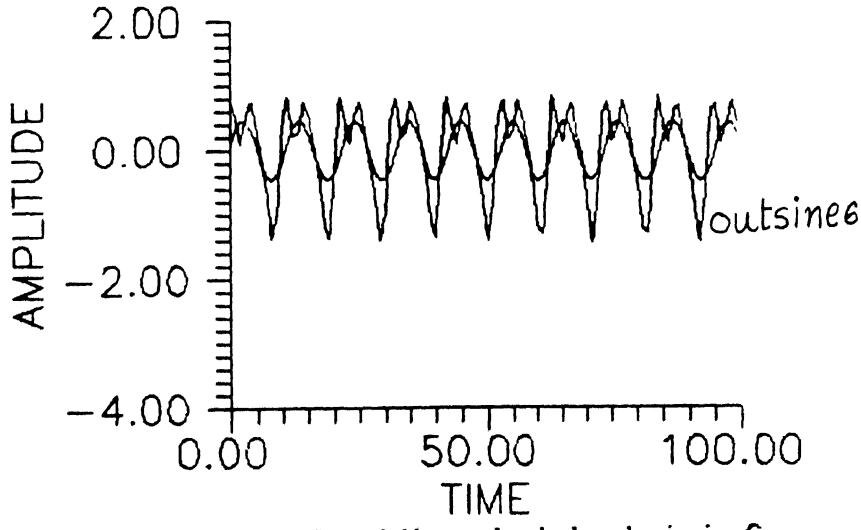


Figure 4.8(c) Target outsine6 and the output due to insine6

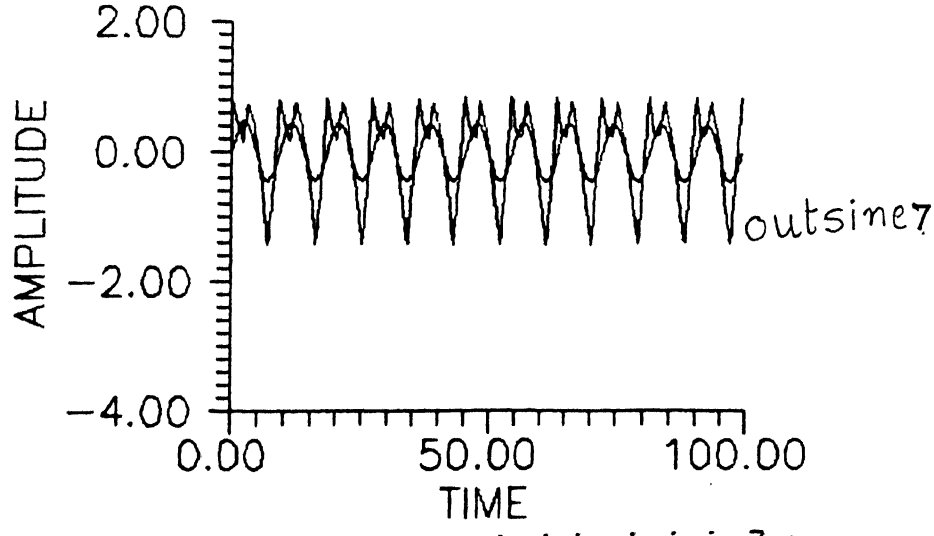


Figure 4.8(d) Target outsine7 and the output due to insine7

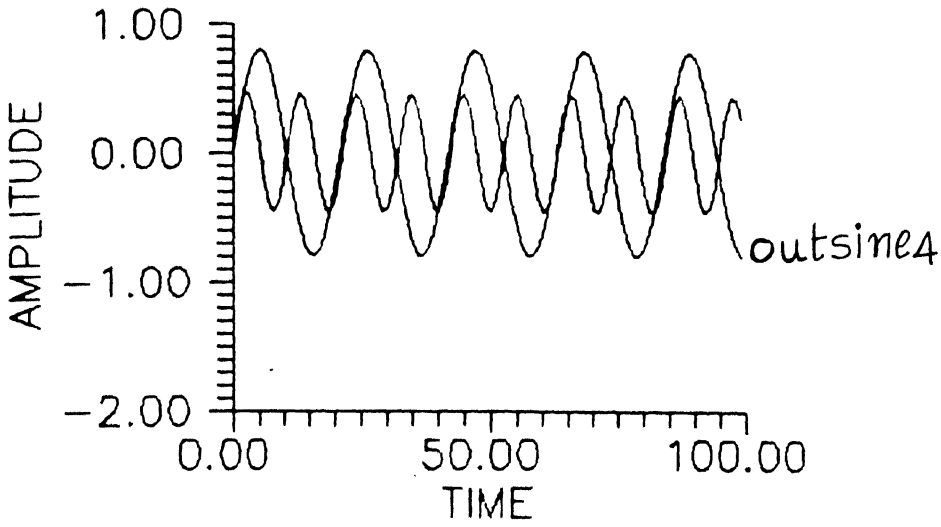


Figure 4.9(a) Target outsine4 and the output due to insine6

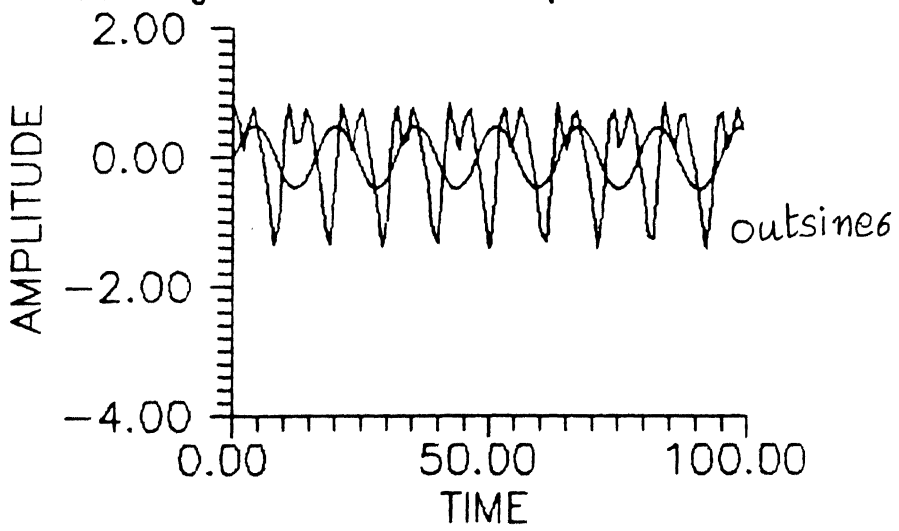


Figure 4.9(b) Target outsine6 and the output due to insine1

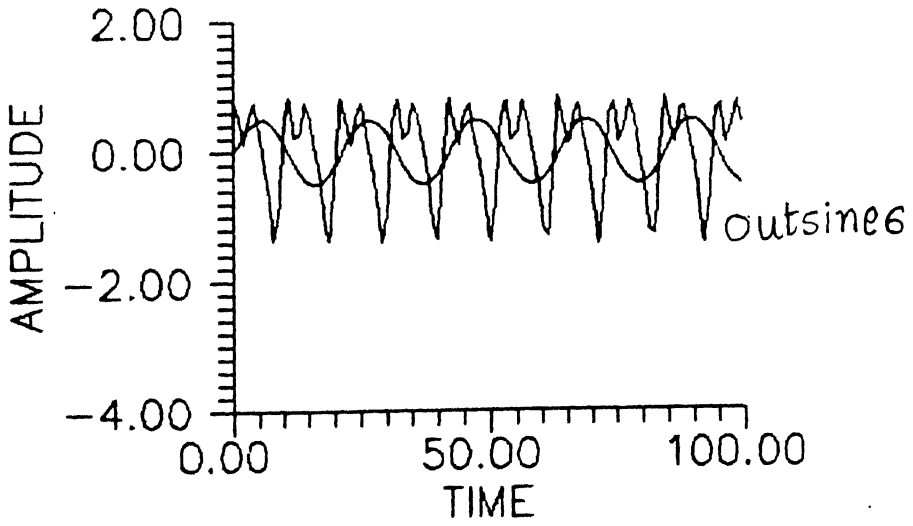


Figure 4.9(c) Target outsine6 and the output due to insine4

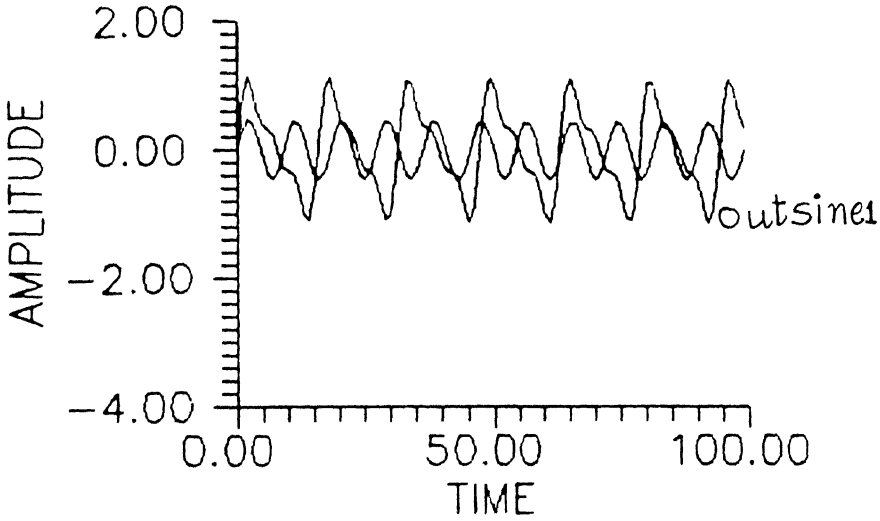


Figure 4.9(d) Target outsine1 and the output due to insine7

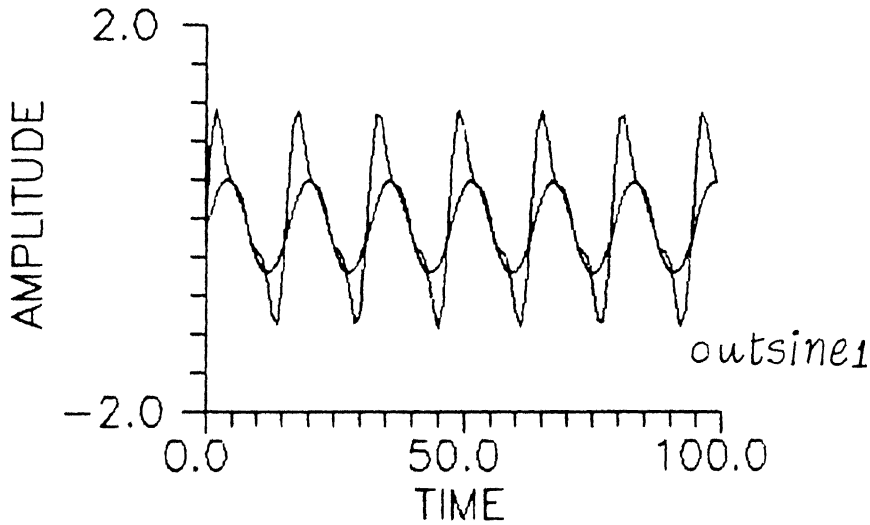


Figure 4.10(a) Target outsine1 and the output due to ibsine1

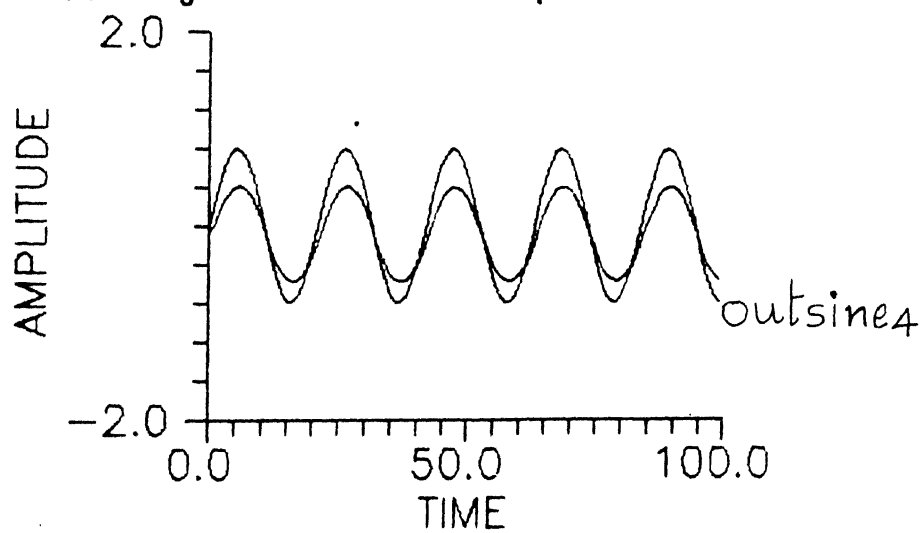


Figure 4.10(b) Target outsine4 and the output due to ibsine4

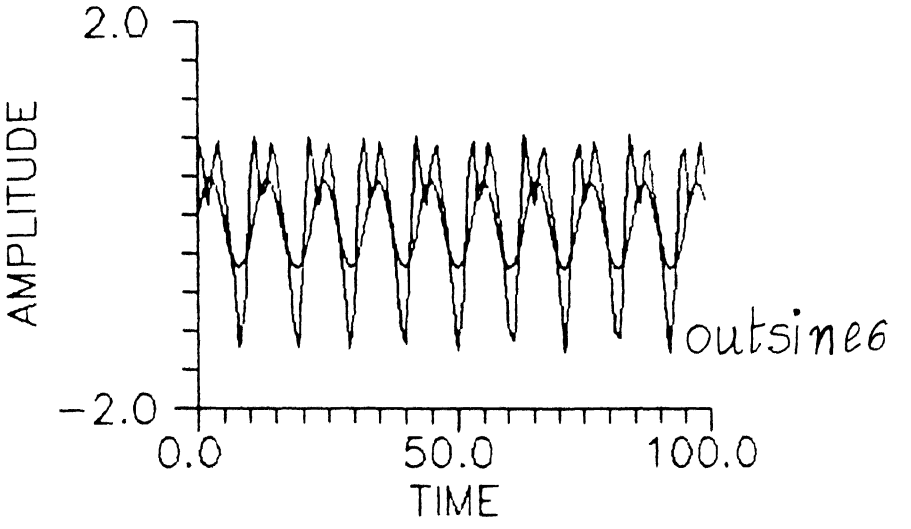


Figure 4.10(c) Target outsine6 and the output due to ibsine6

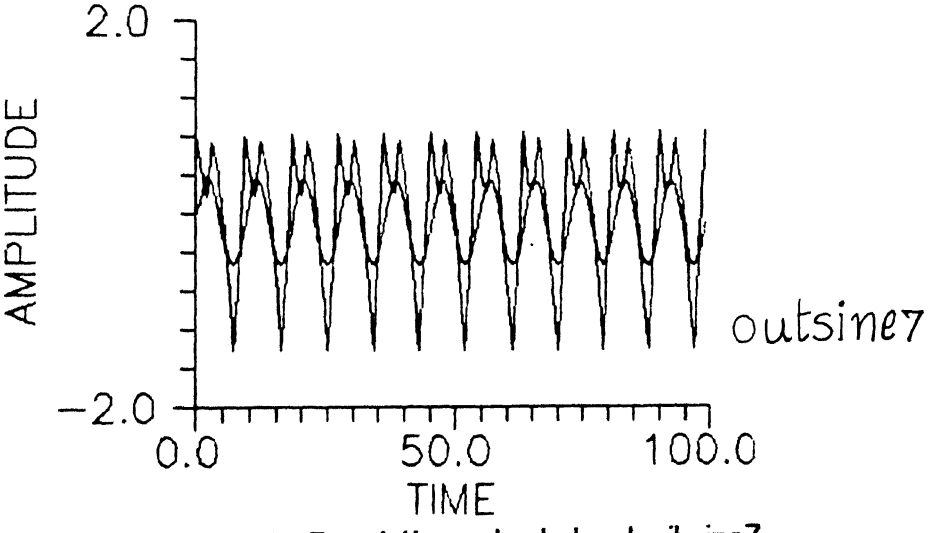


Figure 4.10(d) Target outsine7 and the output due to ibsine7

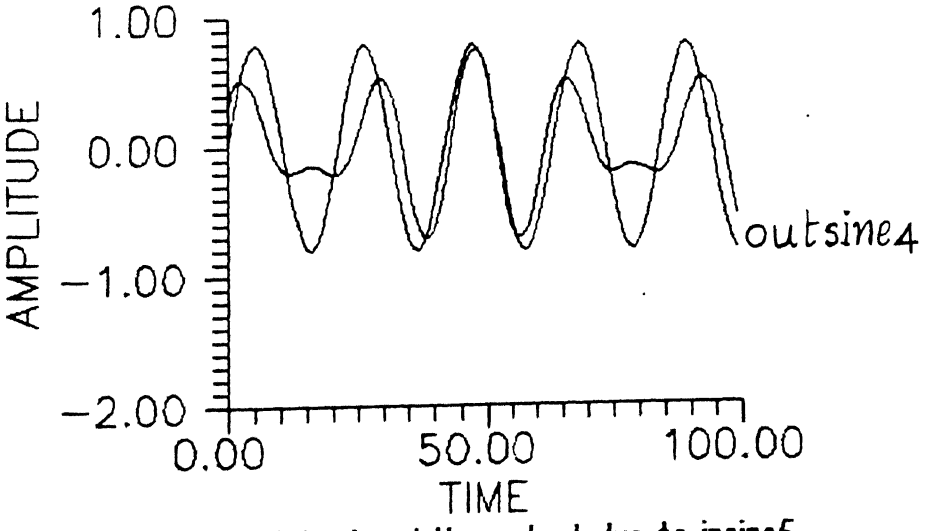


Figure 4.10(e) Target outsine4 and the output due to insine5

4. Experiment 4

In this experiment we investigated into the possibilities of training the network to recognise binary patterns obtained by reversing the bits in the exemplar with a probability 0.1. The three exemplar sequences we chose were periodic sequences of total length 160. Each period had 16 values where the subsequence in a period was one of the rows of a 16x16 Hadamard matrix. The last three rows of the Hadamard matrix were used for the three exemplars respectively. Each of these sequences were used both as inputs and their own targets at the time of training the network. The figures 4.11, 4.12 and 4.13 show one period of each of these sequences h1, h2 and h3 respectively. The corrupted sequences hb1, hb2 and hb3 corresponding to h1, h2 and h3 are shown in figures 4.14, 4.15 and 4.16 respectively. Training was carried out in two stages, first using the learning coefficients in table 4.2(b) and then continued using the ones in table 4.2(d). The weights and filter parameters obtained on training are shown in table 4.3.7 while the results of the experiment are shown in table 4.3.8.

Inputs and targets:

[1] h1 : [1 -1 1 -1 -1 1 -1 1 -1 1 -1 1 1 -1 1 -1]

[2] h2 : [1 1 -1 -1 -1 -1 1 1 -1 -1 1 1 1 1 -1 -1]

[3] h3 : [1 -1 -1 1 -1 1 1 -1 -1 1 1 -1 1 -1 -1 1]

Test inputs :

[1] hb1

[2] hb2

[3] hb3

Error at start = 479.91

Error reduced to 13.29

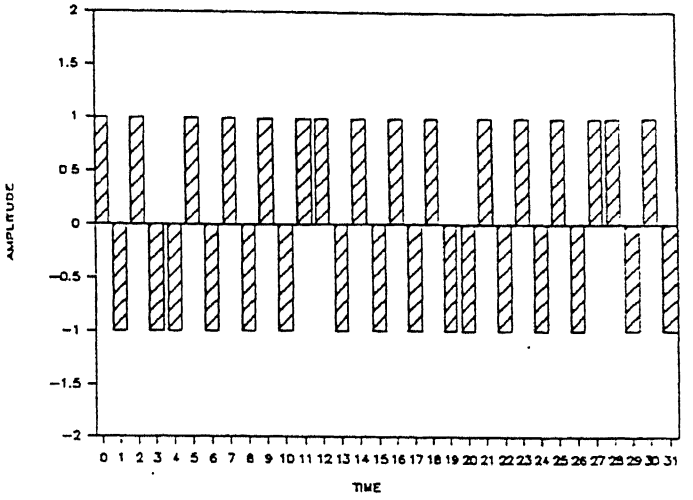


Figure 4.11 Pattern h1 obtained from the 14th row of a 16x16 Hadamard Matrix

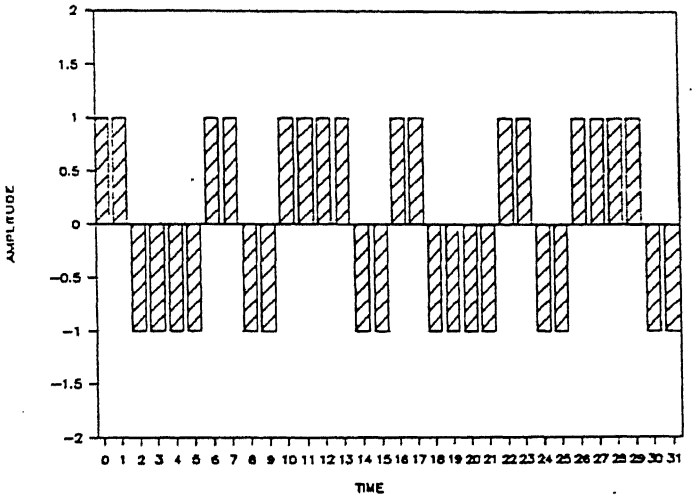


Figure 4.12 Pattern h2 obtained from the 15th row of a 16x16 Hadamard Matrix

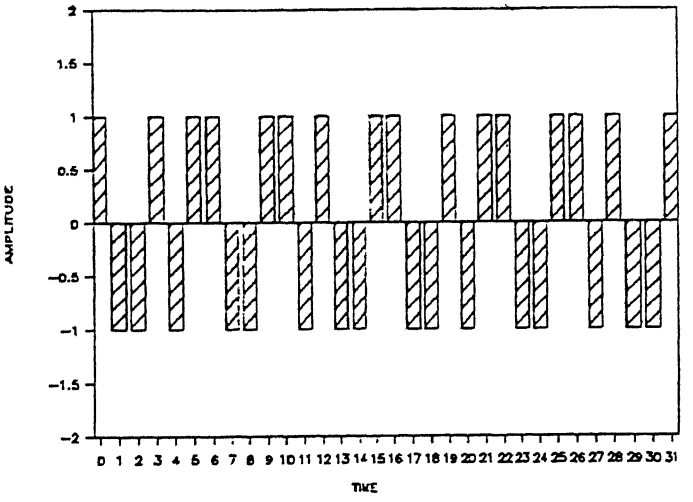


Figure 4.13 Pattern h3 obtained from the 16th row of a 16x16 Hadamard Matrix

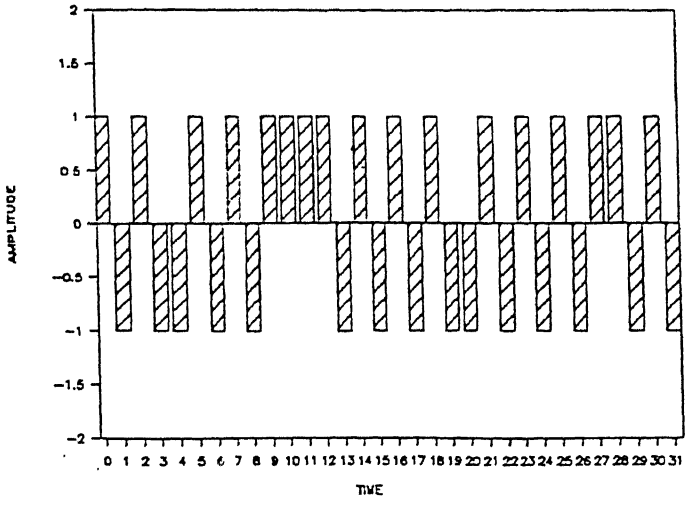


Figure 4.14 Pattern hb1 obtained by corrupting exemplar h1.

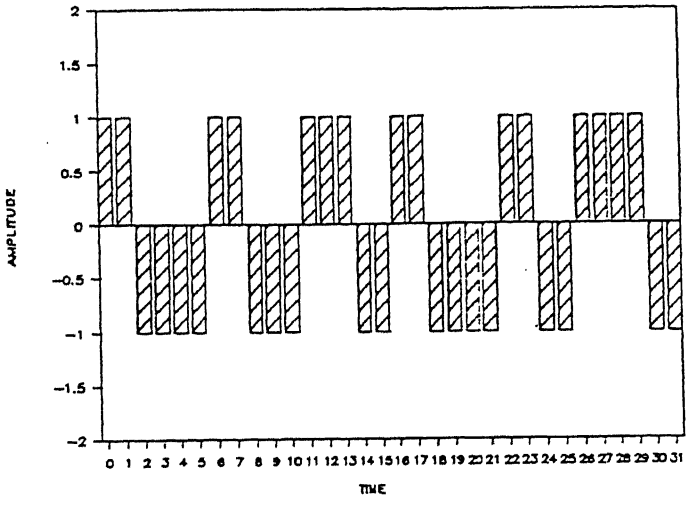


Figure 4.15 Pattern hb2 obtained by corrupting exemplar h2.

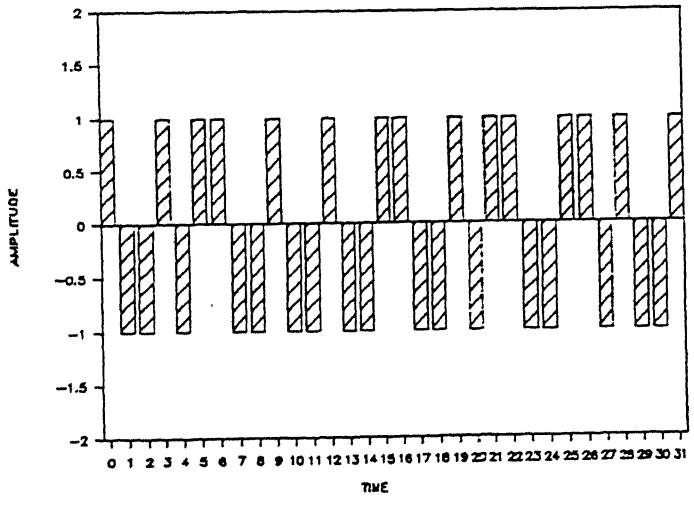


Figure 4.16 Pattern hb3 obtained by corrupting exemplar h3.

The error decreased drastically when learning progressed as seen above. Such a low value of E could be reached probably because the three exemplar sequences are orthogonal to one another. This is a case where target prediction is exceptional. On feeding the input test patterns obtained by corrupting the actual exemplars the net could easily identify the classes to which these belonged to. However, the target prediction was not as good as it was with the original exemplar input patterns, since the error E failed to be within the 10% limit stipulated in the objectives. What is interesting is that the network could actually recognise corrupted patterns obtained by reversing the bits of the exemplar patterns with a *probabilty of as high as* 0.4. The prediction in this case was obviously poor, nevertheless the network could well recognise the corrupted inputs with no ambiguity at all.

Error = 13.293022

	I1	I2	J1	J2	J3	J4	J5	K1
ω	1.59397	1.25129	47.37432	2.10657	1.48322	0.25149	0.83128	2.95072
a	-0.35616	0.16221	0.50000	0.01513	-0.02656	-0.06974	-0.04575	-0.25664

WEIGHTS

w_{il}	4.28524			2.78247				
w_{ji}	0.91606	1.03820		0.28031	0.11213		0.29729	0.14653
	0.25921	0.20018		0.27263	0.14760			
w_{kj}	11.45798		0.07855		0.09317	0.08700	0.08666	

Table 4.3.7 Weights and filter parameters of experiment 4 (section III)

S.no.	Input	Target	Error	Class
1.	h1	h1	5.12	h1
		h2	361.54	
		h3	375.48	
2.	h2	h1	303.92	
		h2	3.50	h2
		h3	261.94	
3.	h3	h1	289.40	
		h2	345.74	
		h3	4.68	h3
4.	hb1	h1	33.66	h1
		h2	370.42	
		h3	355.01	
5.	hb2	h1	335.05	
		h2	46.35	h2
		h3	277.54	
6.	hb3	h1	287.11	
		h2	348.36	
		h3	48.66	h3

Table 4.3.8 Results of experiment 4 (section III)

IV. Spatio-temporal pattern recognition

In the experiments described so far we dealt with a single-input single-output network and performed purely temporal pattern recognition. In the next two experiments that we describe we deal with a 2-input 2-output system which performs spatio-temporal pattern recognition.

1. Experiment 1

In this experiment a network with two input and two output nodes was used. The learning rates and number of neurons in each layer are shown in table 4.4.1.

layer	Neurons	η	η_ω	η_a
4	2	0.1	0.1	0.1
3	5	0.1	0.1	0.1
2	2	0.1	0.1	0.1
1	2	-	-	-

Table 4.4.1 Learning rates used in experiment 1 (section IV)

On the lines of the experiments in section I, we examined whether the network with the number of elements as shown in table 4.4.1, could learn a vector sequence. We chose two sinusoids for the two input sequences that formed a spatio-temporal pattern. The targets corresponding to these two sequences in earlier experiments, now represented a single target pattern in this experiment as shown below.

Input : ins12 = (insine1,insine2)

Target : outs12 = (outsine1,outsine2)

Test inputs :

[1] ibs12 = (ibsine1,ibsine2)

[2] ins13 = (insine1,insine3)

[3] ins21 = (insine2,insine1)

Test target : outs21 = (outsine2,outsine1)

Error at start = 82.11

Error reduced to 27.33

The recognition results are listed in table 4.4.2 and the final weights and filter parameters are shown in table 4.4.3.

S.no.	Input	Target	Error	Class
1.	ins12	outs12	27.33	outs12
2.	ibs12	outs12	30.60	outs12
3.	ins12	outs21	168.64	NOT(outs21)
4.	ins13	outs12	155.64	NOT(outs12)
5.	ins21	outs12	167.03	NOT(outs12)

Table 4.4.2 Results of experiment 1 (section IV)

The entries (1) and (2) in table 4.4.2 show how well the network recognises the training pattern as well as the noise corrupted version of this pattern. Entry (3) shows that the network makes a clearcut distinction between the target outs12 = (outsine1,outsine2) and the test target outs21 = (outsine2,outsine1). Likewise, entry (5) shows that the network clearly discriminates between (insine1,insine2) and (insine2,insine1). Thus, it preserves the spatial ordering in a spatio-temporal pattern. Entry (4) shows how the network can distinguish a pattern foreign to a class.

As far as learning and error-minimisaton are concerned the network behaved in exactly the same manner in which it did in earlier experiments. The next experiment deals with spatio-temporal pattern recognition.

Error = 27.328609

ω	1.59841	1.27803	2.53387	1.83778	1.48736	0.25121	0.83063	2.95354
	2.29808							
a	-0.29532	0.50000	-0.29188	-0.50000	-0.01140	0.12719	0.09032	-0.38728
	-0.06803							

WEIGHTS

w_{jl}	1.08973	1.15624		1.25827	1.26015		
w_{ji}	1.14650	1.14755		1.02910	1.09817	0.17060	0.16107
	0.13689	0.15642		0.15697	0.15222		
w_{kj}	5.33426	4.35246	0.07585	0.05820	0.03893		
	4.24748	3.67675	0.08270	0.04160	0.04636		

Table 4.4.3 Weights and filter parameters of experiment 1 (section IV)

2. Experiment 2

In this experiment we trained three spatio-temporal patterns and carried out recognition tests. The network once again had two inputs and two outputs. Table 4.4.1 and 4.4.4 list the values of the learning coefficients used in two stages of learning. At first the coefficients shown in table 4.4.1 were used to train the net until the network found a deep enough minimum. Then, learning was carried out further using the coefficients in table 4.4.4. The patterns are as follows

Inputs :

[1] ins12 = (insine1,insine2)

[2] ins14 = (insine1,insine4)

[3] ins21 = (insine2,insine1)

Targets :

[1] outs12 = (outsine1,outsine2)

[2] outs14 = (outsine1,outsine4)

[3] outs21 = (outsine2,outsine1)

Test inputs :

[1] ibs12 = (ibsine1,ibsine2)

[2] ibs14 = (ibsine1,ibsine4)

[3] ibs21 = (ibsine2,ibsine1)

[4] ins24 = (insine2,insine4)

[5] ibs24 = (ibsine2,ibsine4)

Error at start = 238.22

Error reduced to 46.07

The results of the experiment are shown in table 4.4.5 and the weights and filter parameters obtained on training are shown in table 4.4.6.

layer	Neurons	η	η_ω	η_a
4	2	0.02	0.01	0.001
3	5	0.05	0.02	0.005
2	2	0.1	0.02	0.008
1	2	-	-	-

Table 4.4.4 Learning rates used in experiment 2 (section IV)

V. Results on robustness

ALFANS was found to be quite robust as illustrated by the results of the experiment we describe now. We trained the network using patterns of experiment 1 in section III. The weights and filter parameters of the trained network are shown

S.no.	Input	Target	Error	Class
1.	ins12	outs12	19.17	outs12
		outs14	106.89	
		outs21	138.09	
2.	ins14	outs12	120.27	
		outs14	6.00	outs14
		outs21	156.92	
3.	ins21	outs12	135.20	
		outs14	139.36	
		outs21	20.90	outs21
4.	ibs12	outs12	22.93	outs12
		outs14	115.94	
		outs21	141.02	
5.	ibs14	outs12	112.08	
		outs14	12.02	outs14
		outs21	151.12	
6.	ibs21	outs12	134.62	
		outs14	145.51	
		outs21	20.90	outs21
7.	ins24	outs12	148.87	
		outs14	34.68	outs14
		outs21	128.06	
8.	ibs24	outs12	140.42	
		outs14	40.45	outs14
		outs21	122.81	

Table 4.4.5 Results of experiment 2 (section IV)

Error = 46.068199

ω	1.56325	2.06059	-0.24026	0.61940	1.48766	0.25044	0.82891	2.95529
	2.29220							
a	0.03812	0.44304	-0.06150	-0.09680	-0.00036	0.17263	0.12463	-0.29172
	-0.05569							

WEIGHTS

w_{ji}	0.99603	1.06230		1.14510	1.15020			
	1.92182	1.68761		1.56612	1.41117		0.19135	0.17231
	0.14850	0.16251		0.16886	0.15905			
w_{kj}	3.79840	2.72963	0.07585	0.05820	0.03893			
	4.91471	4.19924	0.08270	0.04160	0.04636			

Table 4.4.6 Weights and filter parameters of experiment 2 (section IV)

in table 4.5.1. These weights and filter parameters were perturbed by editing them manually. The set of weights and parameters that resulted are shown in table 4.5.2. The network was then employed to recognise the test patterns of experiment 1 of section III. The resulting errors as well as the classification decisions are shown in table 4.5.3 along with the errors that resulted in experiment 1 of section III.

From table 4.5.3 it can been seen that there is virtually no ambiguity in classifying the input even when some of the weights and filter parameters have been perturbed around their original values after training. Only a degradation in performance can be noted, in the sense that the prediction error E

Error = 14.235181

	I1	I2	J1	J2	J3	J4	J5	K1
ω	1.79858	0.28189	91.3500	2.11522	1.49123	0.24862	0.82828	2.90972
a	-0.07537	-0.5000	-0.17635	-0.05913	-0.00642	0.11017	0.06117	0.46169

WEIGHTS

w_{jI}	4.75082			1.27432				
w_{ji}	-1.57750	0.45769		0.45962	0.20140		0.50480	0.24955
	0.45752	0.30029		0.46644	0.24458			
w_{kj}	-13.80250		0.07855	0.09317		0.08700	0.08666	

Table 4.5.1 Weights and filter parameters of experiment 1 of section III.

between the output and the target can be seen to have increased while the error between the ouput and some unrelated target can be seen to have decreased. But clearly, these changes in error E are not large enough to influence decision making and jeopardize the recognizing capabilities of the system.

	I1	I2	J1	J2	J3	J4	J5	K1
ω	1.79058	0.26189	90.5000	2.01522	1.49113	0.24862	0.82828	2.70972
a	-0.07017	-0.47200	-0.17635	-0.06913	-0.00642	0.10017	0.06117	0.46169

WEIGHTS

w_{jl}	4.750			1.37032				
w_{ji}	-1.27750	0.49769		0.45962	0.2000		0.40480	0.24955
	0.35752	0.30029		0.46644	0.24458			
w_{kj}	-10.80250		0.07855	0.09317		0.08700	0.09666	

Table 4.5.2 Weights and filter parameters obtained on perturbation.

S.no.	Input	Target	Error (Section III) (Experiment 1)	Error (Robustness) (Experiment)	Class
1.	insine1	outsine1	9.17	12.63	outsine1
		outsine2	70.59	50.48	
2.	insine2	outsine1	75.42	55.18	
		outsine2	5.07	8.30	outsine2
3.	ibsine1	outsine1	15.01	14.93	outsine1
		outsine2	72.74	50.16	
4.	ibsine2	outsine1	76.56	55.12	
		outsine2	9.86	10.46	outsine2
5.	insine6	outsine1	76.79	55.95	
		outsine2	6.16	8.89	outsine2

Table 4.5.3 Results of experiment on robustness

VI. Results on convergence

Some experiments were performed to examine convergence of the learning algorithm. Empirically, the algorithm can be considered to be convergent if it

leads to stable values of weights and filter parameters; values that change very little when learning is continued further.

In experiment 1 of section III we carried out learning, initially with the rates shown in table 4.2 (b). The error at start was 84.605 which reduced to 17.463 after 114 iterations. Thereafter, error increased drastically. This happened probably, because the weights and parameters had grown to such high values that the input net(n) to the non-linearity in each neural unit had increased so much in magnitude as to cause the neural units to go into saturation, thereby causing the neural outputs to switch between the -1 and +1 limits. Therefore, in the back-propagation phase the weights and parameters did not adjust finely. Instead they changed drastically and made the algorithm show divergent behaviour.

After the 114th learning cycle we reduced the learning coefficients to those shown in table 4.2 (c) and carried on with the learning procedure. The system encountered another minimum and E reduced further down to 14.246 after another 163 iterations and once again the error started increasing. Upon further reducing the learning coefficients to those in table 4.2 (d) the network cruised slowly and the error reduced to 14.235 and varied very little thereafter. Plots of error E versus learning cycles for various cases of learning coefficients in a number of experiments are shown in figures 4.17(a) through (f). The weights and filter parameters for experiment 1 of section III, plotted against learning cycles can be seen in figures 4.18(a) and (b), 4.19 and 4.20. The trend seen here is what has been exhibited in almost all experiments that we have described earlier.

Our conclusion based on the empirical observations we collected is that the network weights and parameters attain stable values at sufficiently low learning rates and that the algorithm displays convergent behaviour. However, if the

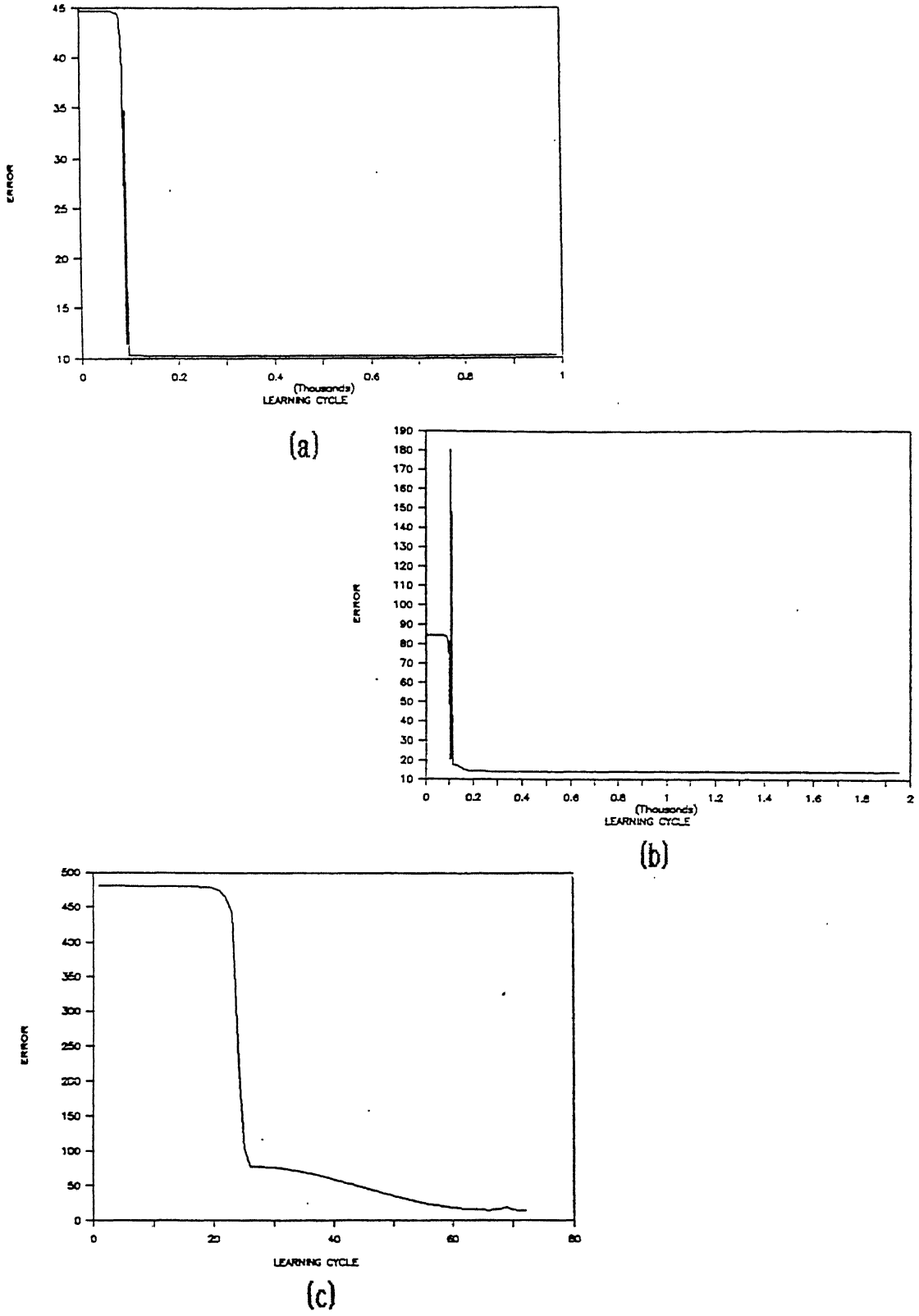
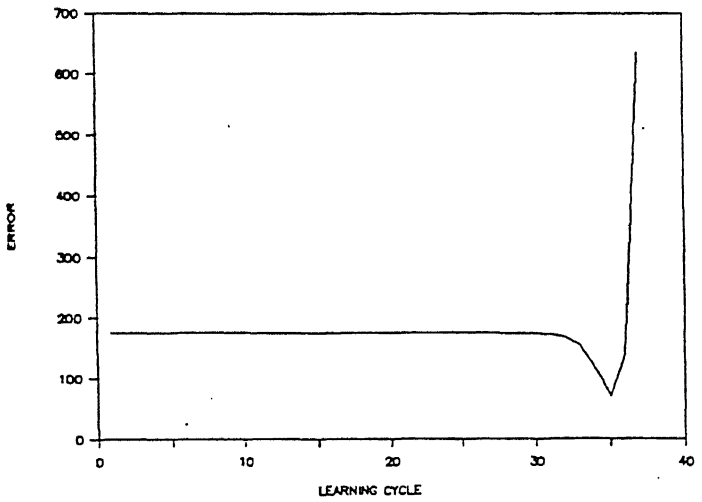
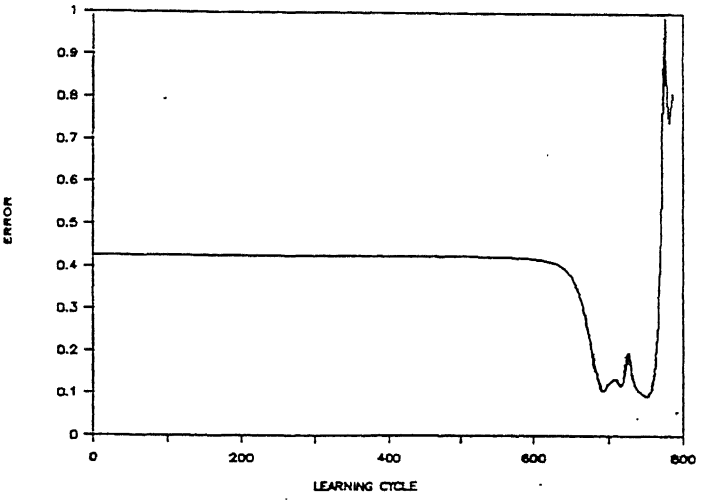


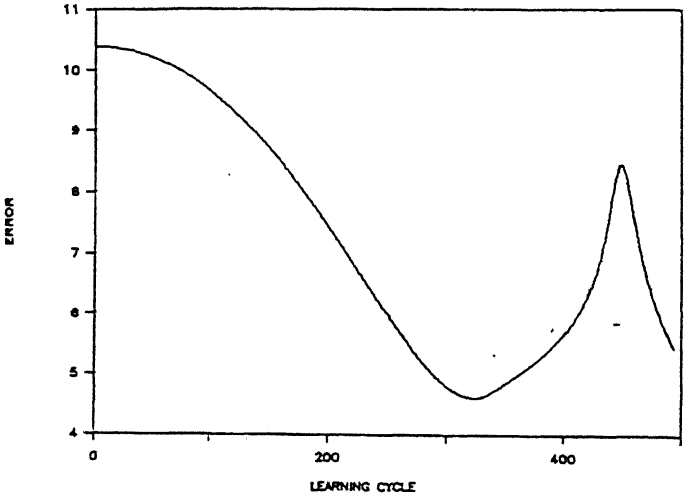
Figure 4.17 Error E vs Learning cycle. (a) [expt.1 sect.II] E varies little at low learning rates. Weights and parameters converge. (b) [expt.1 sect.III] On switching to low rates, weights and parameters converge. (c) [expt.4 sect.III] Error E reduces steadily at low learning rates.



(d)



(e)



(f)

Figure 4.17 Error E vs Learning cycle. (d) [expt.3 sect.III] At high learning rates, error blows up after a minimum. (e) [expt.5 sect.II] At large rates of learning, error blows up after reaching a minimum twice. (f) [sect.VI] Error reaches a new minimum. Perturbed weights and parameters take new values. System is seeking a second minimum at low learning rates.

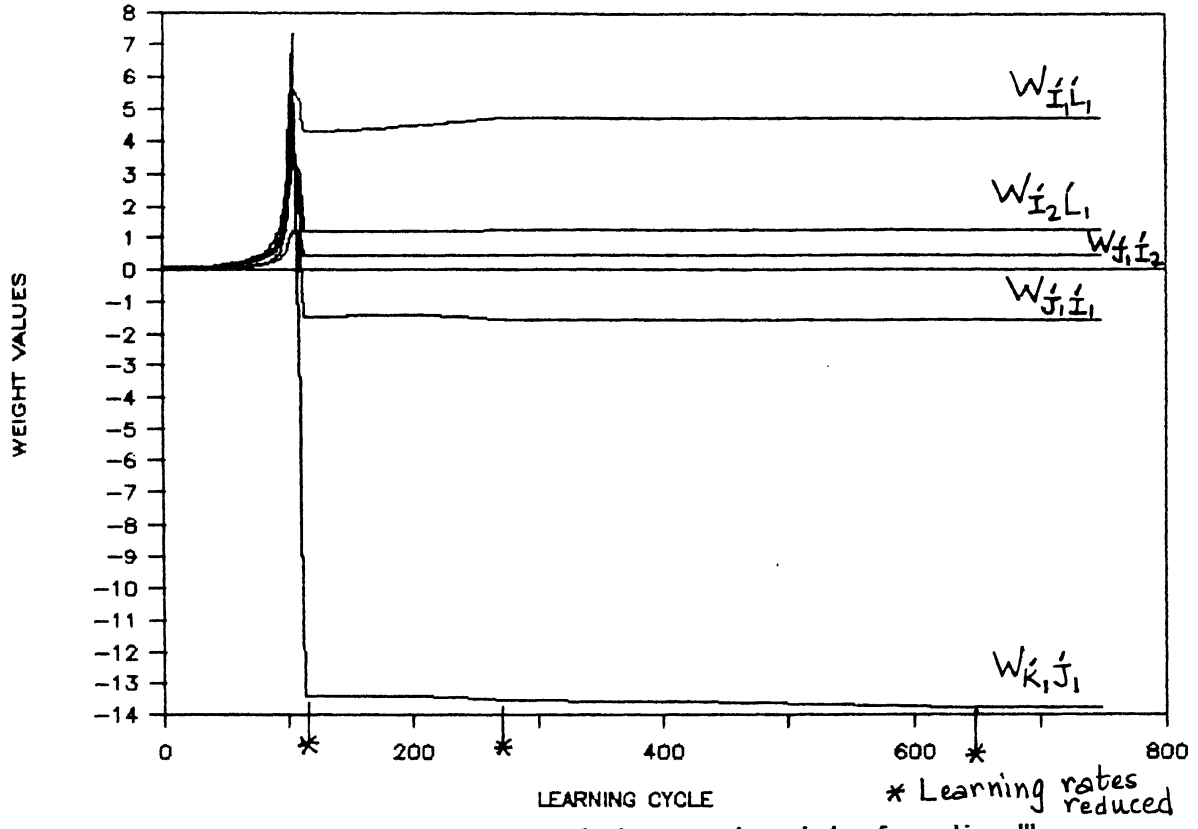


Figure 4.18(a) Weights vs Learning cycle in experiment 1 of section III.

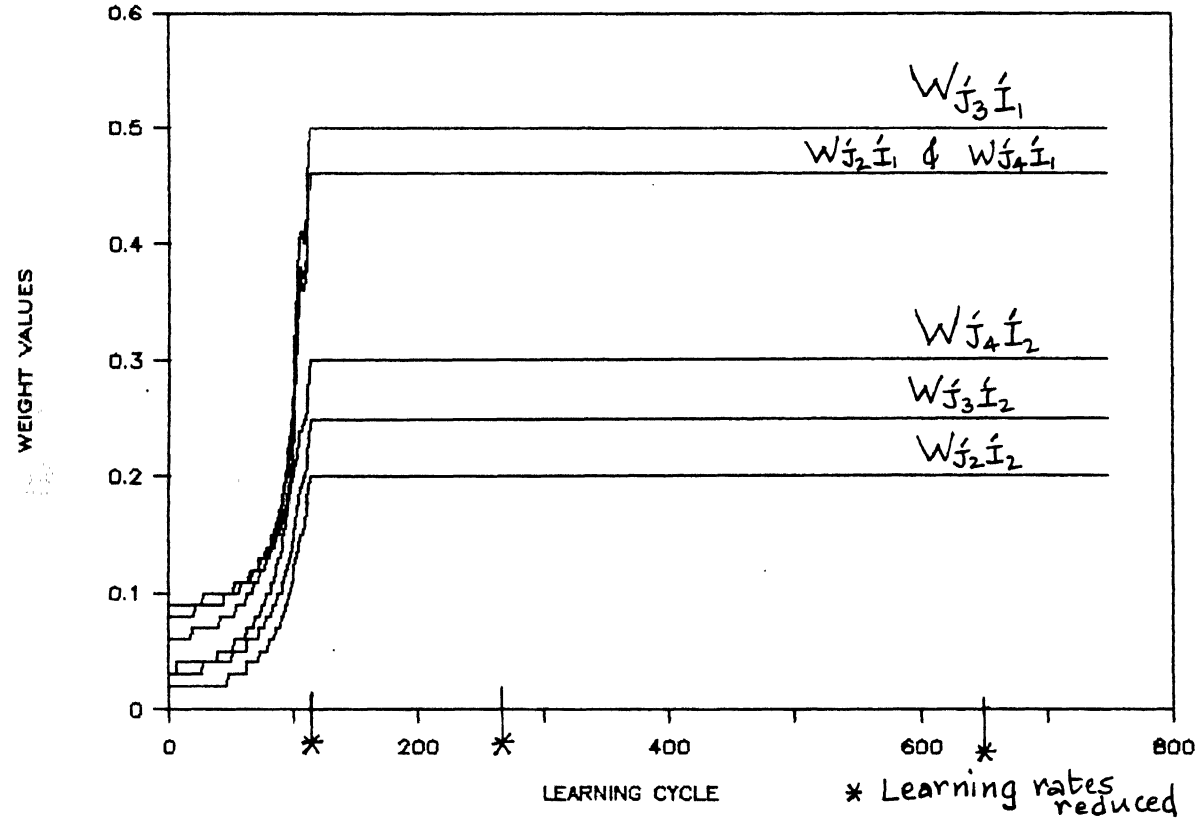


Figure 4.18(b) Weights vs Learning cycle in experiment 1 of section III.

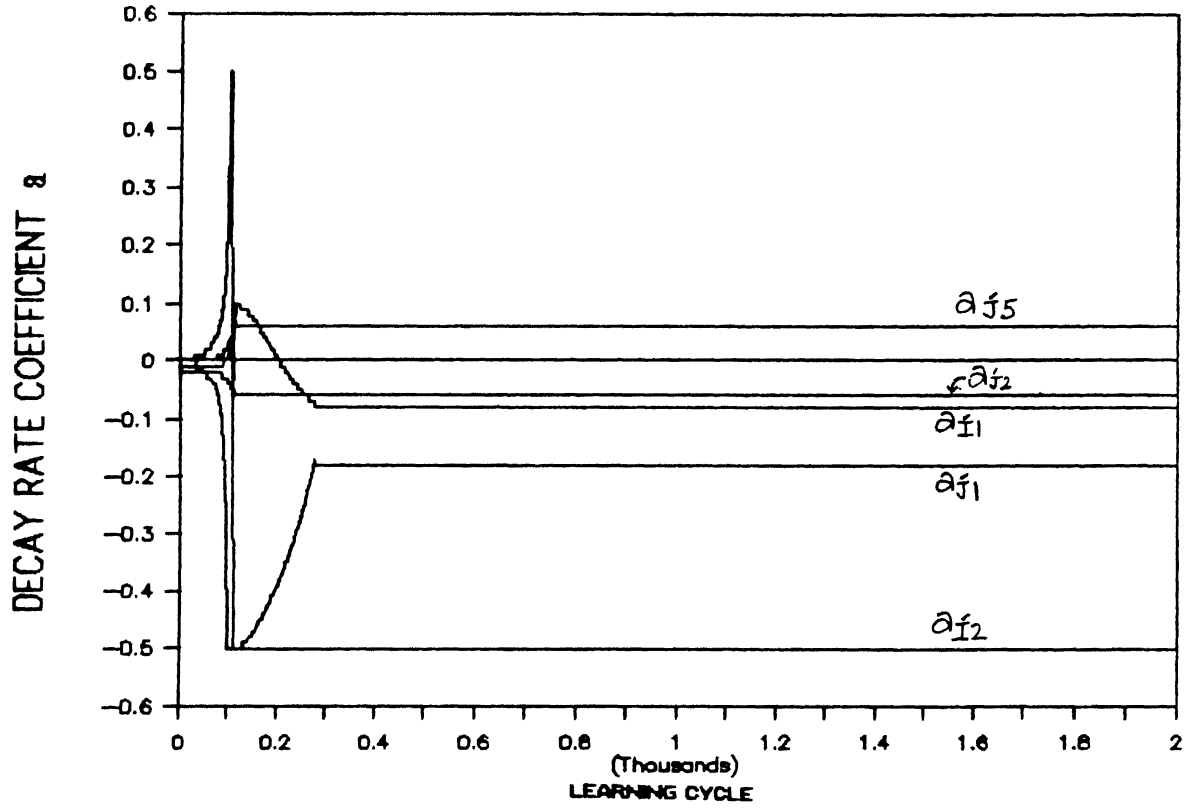


Figure 4.19 The decay rates a vs learning cycle in experiment 1 of section III.

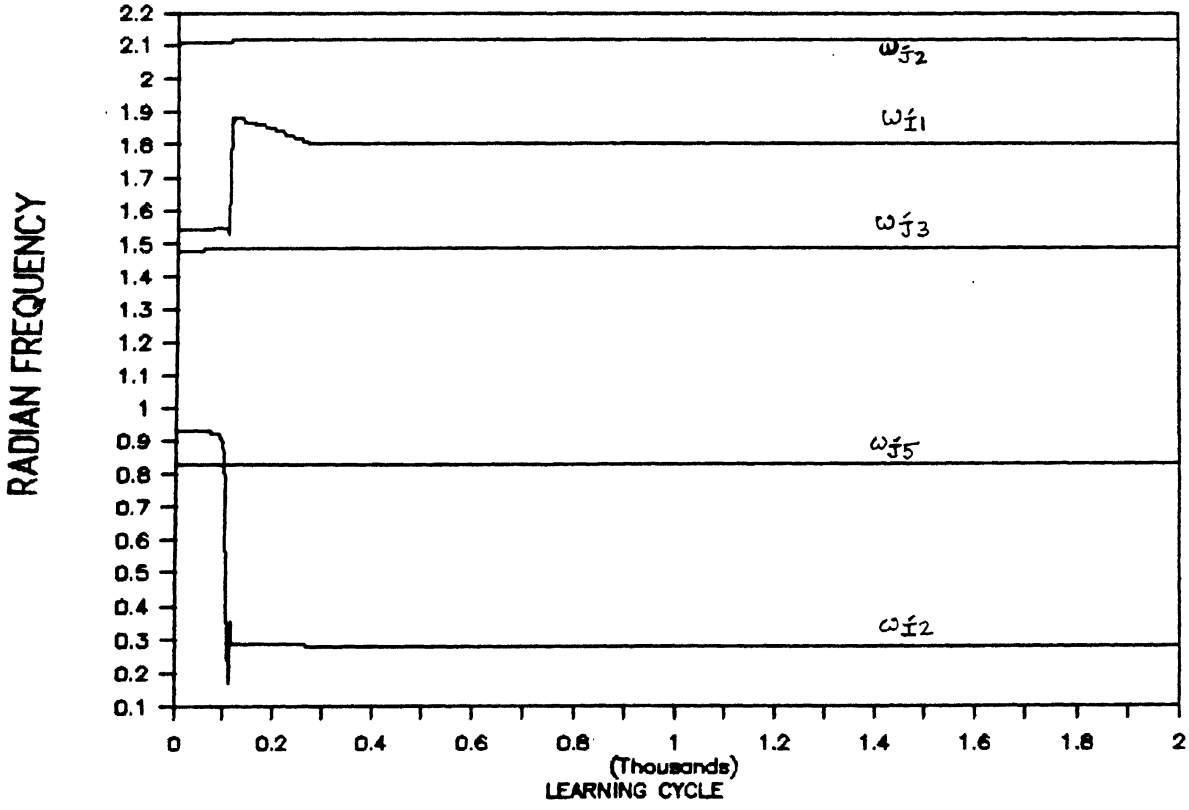


Figure 4.20 The parameters ω vs learning cycle in experiment 1 of section III.

learning coefficients are high then the network exhibits saturaton after going through a deep enough minimum in the error E and the algorithm exhibits divergence.

That the algorithm converges monotonically cannot be true because the system actually jumps from one spurious minimum to another till it lands into a deep enough minimum. If the learning rate coefficients are small then the algorithm does not move the system out of the minimum and the algorithm displays convergent characteristics. What is lacking is any sort of theoretical analysis that can acertain convergence or divergence of this algorithm. This is because of the highly non-linear nature of the neural system.

In another experiment, we used the weights and patterns associated with experiment 1 of section III and fed the inputs insine1 and insine2 to the network. The generated outputs y_1 and y_2 were then made the targets and the error was obviously zero. Next, we used the perturbed weights used in section V (see table 4.5.2) and carried on with the learning process with y_1 and y_2 as the targets. The results obtained are as shown below

Inputs :

[1] insine1 [2] insine2

Targets :

[1] y_1 [2] y_2

Error at start = 10.3804

Error reduced steadily to 4.6239

The system reached an error minimum and the weights and filter parameters stabilised at their new values as shown in table 4.6.1. This clearly shows that the network has multiple choices of weights and parameters which minimise the error E

although at the minimum the error E may not be the same on all occasions. In this particular exrperiment the error could never have reached zero because some initial error always exists in predicting the target pattern at n=0 and this contributes a non-zero part to the total error E.

Error = 4.623922

	I1	I2	J1	J2	J3	J4	J5	K1
ω	1.92461	0.25811	90.70726	2.01551	1.49116	0.24849	0.82800	2.70192
a	-0.16523	-0.50000	-0.48256	-0.06953	-0.00632	0.10128	0.06209	0.45425

WEIGHTS

w_{jl}	4.81934			1.33039				
w_{ji}	-1.19193	0.46377		0.45906	0.20022		0.40415	0.24980
	0.35686	0.30055		0.46575	0.24485			
w_{kj}	-10.71904		0.07855	0.09317		0.08700	0.09666	

Table 4.6.1 New values of weights and filter parameters obtained in the above experiment.

With these results we conclude this chapter, having presented a considerable number of cases to bring out the various idiosynchracies of the network and the associated learning algorithm alongwith the results that make evident the capacity of the system to perform spatio-temporal pattern recognition.

Chapter 5
Conclusion and Discussion

In this thesis various aspects of spatio-temporal pattern recognition have been considered along with discussion on some approaches that researchers have taken to perform this task. Chapter 2 discusses some of the networks that researchers have proposed for speech recognition and text to phoneme transcription. This chapter also includes a discussion on outstar based networks which can be used for generating complex spatio-temporal patterns useful in tasks such as motion control of robots. There is also a discussion on how self-organising networks can be used for speech segmentation and recognition.

Some ideas that led to the design of a feed-forward back-propagation network for spatio-temporal pattern recognition have been presented. Based on these ideas a linear filter artificial neuron system has been proposed in chapter 3 and a learning algorithm has been derived to train this network. Conditions on the stability of various filters used in this network have also been noted. Results of computer simulations to test this network have been presented in chapter 4.

I. Comments and suggestions for future work

There are some critical comments to be made on ALFANS. In this system the output neural units perform some kind of signal synthesis. The inputs to these units are sinusoids or decaying sinusoids, or some such signals and the weights connecting the output units to the previous layer, act like the synthesis

coefficients. The output signal so synthesized, is expected to predict the target with fair accuracy. Based on our simulations we observe that the prediction performance of the network is not upto the mark in most cases. One possible suggestion to improve the performance is to remove the filters in the output units and retain the standard neuron that is normally used in feed forward networks for static pattern recognition. This essentially amounts to chosing an output filter with a unit impulse for its impulse response. This way the low pass nature of the output filter that is now diminishing the high frequency content of the output signal will be eliminated. An increase in high frequency content of the synthesized output is very likely to enhance the perfomance of the network in predicting the target waveform.

One other suggestion is that an empirical study of this network be carried out using various kinds of linear filters in order to examine as to what type of filters are most appropriate for a given set of patterns to be trained. For instance, it should seem appropriate to use first order filters to train decaying exponentials, while a resonator could possibly enhance the network's performance on recognition of sinusoids.

There is an important observation to make regarding the learning rate coefficients. It is very clear that the algorithm is not convergent for large values of learning rate coefficients although it is seen to converge for small rates. However, large rates allow the system to quickly bypass local minima and descend along a deep minimum, while small values of learning coefficients cause the system to glide slowly and increase the possibility of the system getting stuck in a spurious local minimum. To get the advantages of these two cases while eliminating their disavantages one could consider the possibility of reducing learning rates

based on the decrease in error from the starting value. For example one could consider making

$$\eta^{next} \propto \frac{1}{\left(E^{start} - E^{current}\right)^{a}} \qquad (5.1)$$

where E^{start} is the error at start, $E^{current}$ is the error after the current iteration, and η^{next} is the value of η for the next iteration.

Another option is to use low value learning rate coefficients and include momentum terms when calculating the increments. This has been highly successful in case of the standard back-propagation network. It is expected that addition of the momentum term will increase the pace of learning and bypass spurious minima even in this algorithm. This has to be examined by carrying out some simulation experiments.

In this thesis we have dealt with sinusoidal as well as binary patterns. One could possibly take up speech recognition using this system in a future endeavour. A short-time Fourier transform can be treated as a spatio-temporal pattern and speech recognition based on this can be performed. One could also feed sampled speech directly to this system without any pre-processing and train the network to perform temporal pattern recognition. One other application domain for ALFANS, besides speech processing, is robotics where it can be used to recognise spatio-temporal signals generated by the sensory apparatus of a robot. Sensing moving pictures and temporal changes in tactical stimulii are some important areas where ALFANS can possibly, be employed.

The present scenario in pattern recognition, control and robotics indicates that much work needs to be done in spatio-temporal pattern analysis and recognition, and this thesis is being concluded with the view that neural networks will become the means to perform this task with efficiency, reliability and ease.

Appendix A
Jury's Test

Jury's test states the conditions under which a filter $H(z) = \frac{N(z)}{D(z)}$, with

$$D(z) = \sum_{i=0}^{N} b_i z^{n-i} \qquad (A.1)$$

is stable.

Using the coefficients of D(z) we construct the following array

Jury's Table

ROW				
1	b_0	b_1	. . .	b_N
	b_N	b_{N-1}	. . .	b_0
2	c_0	c_1	. . .	c_{N-1}
	c_{N-1}	c_{N-2}	. . .	c_0
3	d_0	d_1	. . .	d_{N-2}
	d_{N-2}	d_{N-3}	. . .	d_0
. . .	. . .	. . .	. . .	. . .
2N-3	r_0	r_1	r_2	

where

$$c_i = \begin{vmatrix} b_0 & b_{N-i} \\ b_N & b_i \end{vmatrix} \qquad \text{for } i=0 \text{ to } N-1$$

$$d_i = \begin{vmatrix} c_0 & c_{N-1-i} \\ c_{N-i} & c_i \end{vmatrix} \qquad \text{for } i=0 \text{ to } N-2 \qquad (A.2)$$

and so on until the row 2N-3 which has just 3 elements in it. Jury's stability criterion then, states that the filter H(z) is stable iff the following are satisfied.

(i) $D(1) > 0$ (A.3.1)

(ii) $(-1)^N D(-1) > 0$ (A.3.2)

(iii) $b_0 > |b_N|$, (A.3.3)

$c_0 > |c_{N-1}|$,

$d_0 > |d_{N-2}|$,

. . .

$r_0 > |r_2|$

Appendix B
Instar and Outstar

I. Introduction

Instar and Outstar configuration along with the associated training rules were first proposed by Grossberg [16]. The instar, as shown in fig. B.1, is a neurode fed by a number of inputs taken through set of synaptic weights. It also has a triggering output which may be connected to other neurodes. The outstar, as shown in fig. B.2 is a neurode with many outputs each of which is driven through a synaptic weight. It has a triggering input which allows the other neurodes to trigger it. In fact in any neural network, every neurode can be looked upon as the focus of an instar and an outstar since each neurode has many inputs and at the same time its output fans out to many other neurodes. Thus, a neural network can be considered to be a vast array of neurodes interconnected amongst themselves, with the inwardly feeding inputs of each instar arising from the outwardly directed outputs of some outstars.

II. Instar training rule

An instar is trained to respond to a specific input vector **X** only. Training proceeds in a way so as to adjust the input weight vector to be like the specified input vector **X**. The instar output is calculated to be the linear combination of the inputs. Thus, one can see that the calculation produces the dot product of the weight vector with an input vector applied at a given time which is a measure of similarity between the two, in case both of them are normalized vectors. The training rule is such that in the testing phase a trained instar comes to respond

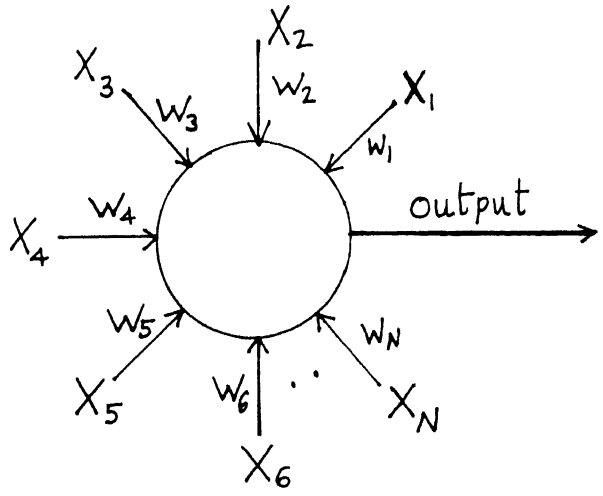


Figure B.1 The Instar.

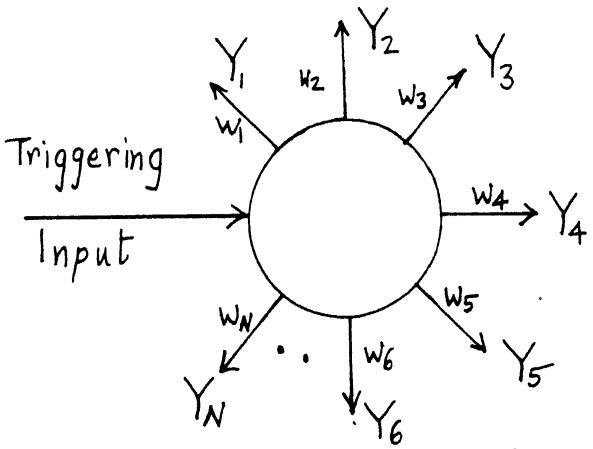


Figure B.2 The Outstar.

most strongly to the input pattern for which it was trained. The weight adjustment rule is given below.

$$w_j^{t+1} = w_j^t + \alpha[x_j - w_j^t] \qquad (B.1)$$

where $w_j t$ is the weight from input x_j after learning iteration t, and α is the training rate coefficient which is chosen to be around 0.1 and is reduced gradually as training progresses.

III. Outstar training rule

While the instar performs recognition the outstar generates the desired excitation pattern to the other neurons when it fires. The outstar training rule adjusts the synaptic weights to be like a desired target vector. The training rule is as follows.

$$w_j^{t+1} = w_j^t + \beta[y_j - w_j^t] \qquad (B.2)$$

where $w_j t$ is the weight at the output y_j after learning iteration t, and β is the training rate coefficient which is chosen to be around 1 and is gradually reduced to zero as training progresses.

Both instar and outstar are trained over a set of training vectors belonging to a class so that the instar weights adjust themselves to represent an average (in some sense) of all the patterns belonging to that class and the outstar can converge to the exemplar pattern when the network is presented with distorted patterns. By including a non-linearity in series with the outstar outputs one can actually train the system to recognise and generate more than just one class of patterns. The instar and outstar are discussed in the references [11], [16] and [54].

Appendix C
Neocognitron

I. Introduction

Neocognitron is a self-organising network proposed by Fukushima and Miyake [14], [15], which is tolerant of deformations as well as shifts in position of the impressed input pattern. This network belongs to a class of competitive learning networks and is an advancement over an earlier version called cognitron which was not shift-tolerant. Repetitive presentation of the input patterns causes the network to self-organise and after sufficient progress the network acquires the ability to classify and correctly recognise these patterns or their variants by perceiving the differences in their shapes.

II. Cells in the neocognitron

The neocognitron is heirarchically organised as shown in figure C.1. It has a number of modules with the innermost one composed of "gnostic cells". Each module has four layers, each carrying a different type of cell or neuron. Two of these layers, namely **Us** and **Uc** carrying the **S-cells** (Us cells) and the **C-cells** (Uc cells) respectively, are further divided into a number of **planes** each of which is an array of cells. The other two layers carry the **Vs** and **Vc** type cells respectively.

The input-output description of these cells follows. Let u(n), n=1 to N be the excitatory inputs and v be the inhibitory input to an S-cell. The output w

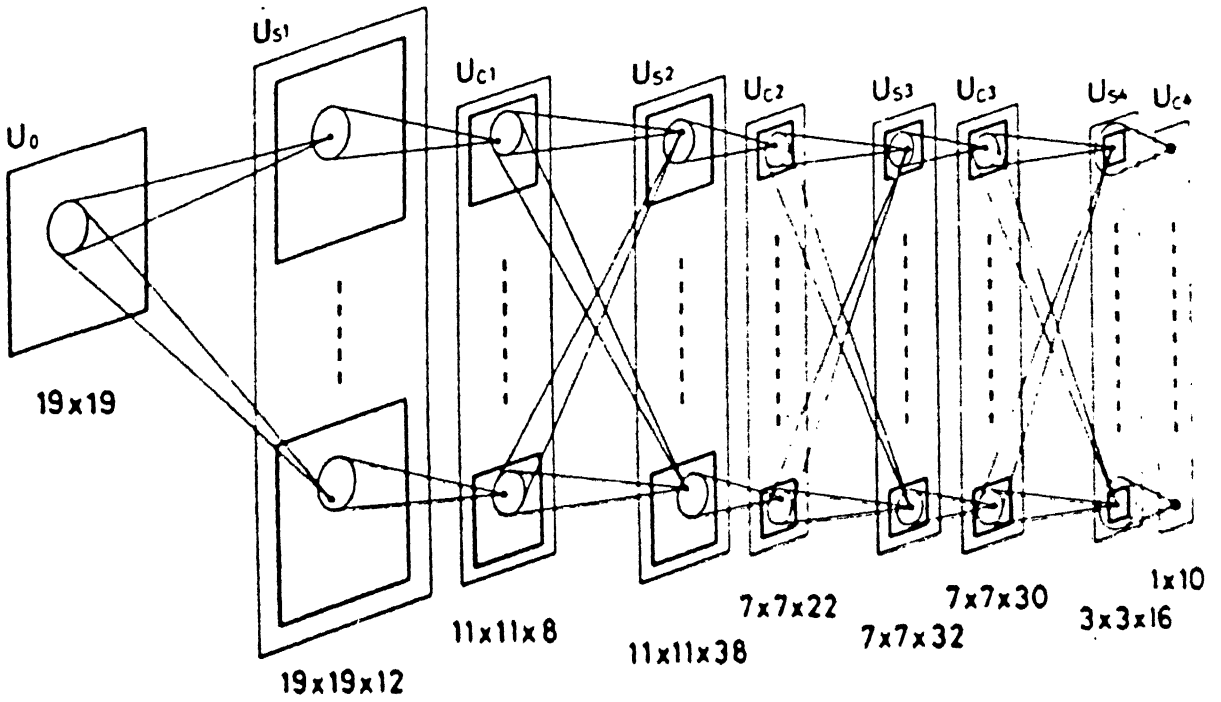


Figure C.1 Schematic diagram illustrating the interconnections between layers in the neocognitron (Reproduced from [14]).

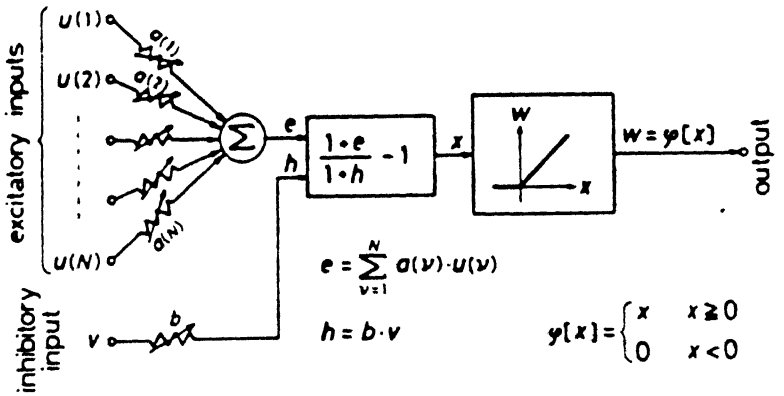


Figure C.2 Input-to-output characteristics of an S-cell: A typical example of the cells employed in the neocognitron (Reproduced from [14])

of the S-cell is defined by

$$w = \varphi\left[\frac{1+\sum_{n=1}^{N} a(n)u(n)}{1+bv} - 1\right] \qquad (C.1)$$

where a(n) and b represent the excitatory and inhibitory interconnection coefficients respectively and the function $\varphi[\ ]$ is given by

$$\varphi[x] = \begin{cases} x & (x \geq 0) \\ 0 & (x < 0) \end{cases} \qquad (C.2)$$

The output of a C-cell is quite similar except that it uses a $\psi[\ ]$ function instead of $\varphi[\ ]$, and is given by

$$\psi[x] = \begin{cases} \frac{x}{\alpha+x} & (x \geq 0) \\ 0 & (x < 0) \end{cases} \qquad (C.3)$$

The parameter α is a positive constant which determines the degree of saturation of the output. The Vc cell has inputs u(n), n=1 to N and its output is given by

$$w = \sqrt{\sum_{n=1}^{N} c(n)u(n)} \qquad (C.4)$$

The Vs-cell input-output relation is quite similar to that of the Vc-cell. In the neocognitron the only the S-cell connection strengths a( ) and b are allowed to adjust during self-organisation.

III. Neocognitron structure

Now we describe the process of updating the cells in the testing phase. The notation $u_{sl}(k_l, n)$ will be used to represent the output of an S-cell in the k_l-th S-plane in the l-th module, where n is the position vector of the cell in that

plane. The C-cell output is denoted in a similar manner. The input is presented to the Uc-cells of the C-layer of the module number 0. The Uc cells of this module are first updated. Then the Vc cells of that module are updated. The updated outputs from these Uc and Vc cells now serve as inputs for updating the Us cells of the next module. The outputs from these Us cells are used as inputs to update the Vs cells of the same module. The updated outputs from the Us and Vs cells of that module are used as inputs to the Uc cells of the module in the same manner in which the input picture is impressed on the Uc cells of module number 0. The Uc cell outputs are used to update the Vc cell outputs of that module which serve as inputs to the next module. This is carried out sequentially from the outermost module, namely module 0 upto the innermost module where the Uc cells are arranged in K planes each holding exactly one cell. These cells are representative of the gnostic cells in the human brain. Each of these gnostic cells represents exactly one class of input patterns and fires when a member of that class is impressed on the input layer. The constitution of a class is decided entirely by the structure and number of cells in the network, the scheme for self-organisation, the threshold parameters and learning rates, as well as, the initial choice of weights which are to be chosen randomly subject to some constraints. Thus, the maximum number of classes of input patterns can be only as many as the number of gnostic cells.

The update equations are as follows. The inhibitory cell $v_{cl-1}(n)$ output is given by

$$v_{cl-1}(n) = \sqrt{\sum_{k_{l-1}=1}^{K_{l-1}} \sum_{p \in S_l} c_{l-1}(p) u^2_{cl-1}(k_{l-1}, n+p)} \qquad (C.5)$$

The values of the fixed interconnection weights $c_{l-1}(p)$ are chosen so as to decrease monotonically with respect to $|p|$ and to satisfy

$$\sum_{k_{l-1}=1}^{K_{l-1}} \sum_{p \in S_l} c_{l-1}(p) = 1 \qquad (C.6)$$

The size of the connecting area S_l is chosen small in the first module and increases as the depth l is increased. The output of a Us cell of the k_l-th S-plane in module l is given by

$$u_{Sl}(k_l,n) = r_l \varphi\left[\frac{1 + \sum_{k_{l-1}=1}^{K_{l-1}} \sum_{p \in S_l} a_l(k_{l-1},p,k_l) u_{Cl-1}(k_{l-1},n+p)}{1 + \frac{r_l}{1+r_l} b_l(k_l) v_{Cl-1}(n)} - 1 \right] \qquad (C.7)$$

where $\varphi[\ ]$ is as defined in (C.2). In case of $l=1$ $u_{Cl-1}(k_{l-1},n)$ stands for $u_0(n)$ and $K_0=1$. Parameter r_l in (C.7) controls the intensity of inhibition. The Vs cells are updated next according to the following rule

$$v_{Sl}(n) = \frac{1}{K_l} \sum_{k_l=1}^{K_l} \sum_{p \in D_l} d_l(p) u_{Sl-1}(k_l,n+p) \qquad (C.8)$$

The Uc cells are the last to be updated and their input-output relationship is described by

$$u_{Cl}(k_l,n) = \psi\left[\frac{1 + \sum_{p \in D_l} d_l(p)\, u_{Sl-1}(k_l,n+p)}{1 + v_{Sl}(n)} - 1 \right] \qquad (C.9)$$

where $\psi[\ ]$ is as defined in (C.3). The fixed interconnection strengths $d_l(p)$ are chosen exactly like $c_l(p)$ and once again the connecting area D_l is set small in the first module and is increased in the inner layers.

IV. Self-organisation in neocognitron

In the neocognitron self-organisation is via unsupervised learning in which the network is repeatedly presented with patterns at the input. After a pattern presentation the cells in all the modules are updated according to the rules specified in the section III. Then, the variable interconnections a( ) and b are adjusted as per the following scheme. Firstly, several "representative" S-cells are chosen from among all the S-cells that yield a large output after being updated but the number of representatives is so restricted that more than one must not be chosen from any single S-plane in a module. The details of the procedure for choosing the representative cells are given in [14]. For a representative S-cell, only those input interconnections which see a non-zero incoming signal are reinforced so that the cell becomes selectively responsive to the stimulus feature that is seen then. All the other S-cells in the S-plane, from which the representative is chosen get their input interconnection strengths reinforced by the same amounts as those for their representative. Let cell $u_{Sl}(\hat{k}_l,\hat{n})$ be a representative. Then, the incoming interconnection strengths are reinforced according to the rule given below

$$\Delta a_l(k_{l-1},p,\hat{k}_l) = q_l c_{l-1}(p) u_{Cl-1}(k_{l-1},\hat{n}+p) \qquad (C.10)$$

$$\Delta b_l(\hat{k}_l) = q_l v_{Cl-1}(\hat{n}) \qquad (C.11)$$

where q_l is a positive constant that determines the speed of increment. The cells in the S-plane from which no representative comes forth do not have their input interconnections varied at all.

An explanation for the specific input-output relations for various cell types and the updating rules for interconnection strengths mentioned above along with an extensive description of the neocognitron behaviour are contained in the original paper by Fukushima and Miyake [14] published in 1982.

Bibliography

Some of the sources contained herein could not be consulted because of their inavailability. Nevertheless, they have been included in this bibliography because they have been cited in the articles that were referred to by the author and were considered significant enough to be listed here for the sake of completeness.

[1] Alexander,I. and Morton,H., *An introduction to neural computing*. London: Chapman and Hall, 1990.

[2] Almeida,L.B. and Wellekens,C.J., ed., *Neural Networks: EURASIP workshop* 1990, Sesimbra, Portugal. Springer-Verlag, 1990. Lecture notes in computer science, vol. 412.

[3] Amit,D.J., *Modeling brain function: the world of attractor neural networks*. Cambridge: Cambridge Univ. Press, 1989.

[4] Arbib,M.A. and Amari,S., ed., *Dynamic interactions in neural network models and data*. New York: Springer-Verlag, 1989. Research notes in neural computing, No.1.

[5] Burr,D.J., "A neural network digit recognizer," *Proc. IEEE Intl. Conf. Syst. Cybern.*, Atlanta, GA, Oct. 1986.

[6] Burr,D.J., "Experiments on neural net recognition of spoken and written text," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, no. 7, pp. 1162-1168, July 1988.

[7] Carpenter,G.A. and Grossberg,S., "Neural dynamics of adaptive pattern recognition: Priming, search, attention, and category formation, *Soc. Neurosci. Abstracts*, vol. 11, pp. 1110, 1985.

[8] Carpenter,G.A. and Grossberg,S., "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Compu., Vision, Graphics, Image Processing*, vol. 37, pp. 54-115, 1987.

[9] Carpenter,G.A. and Grossberg,S., "ART2: Self-organisation of stable category recognition codes for analog input patterns," *Applied Optics*, vol. 26, pp. 4919-4930, 1987.

[10] Casar-Corredera,J.R., Alcázar,J.M. and Figueiras-vidal,A.R., "Some simple non-linear prediction schemes," *Digital Signal Processing* - 84, Elsevier Sci. Publ. (North-Holland) 1984.

[11] Caudill,M. and Butler,C., *Naturally intelligent systems*. Cambridge, MA: MIT Press, 1990.

[12] Denker,J.S., ed., *Neural networks for computing: AIP conference proceedings, Snowbird, Utah*, 1986. New York: American Institute of Physics, 1986.

[13] Fukushima,K., "Cognitron: A self-organising multilayered neural network," *Biological Cybernetics*, vol. 20, pp. 121-136, 1975.

[14] Fukushima,K. and Miyake,S., "Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position," *Pattern Recognition*, vol. 15, no. 6, pp. 455 - 469, 1982.

[15] Fukushima,K., Miyake,S. and Takayuki,I., "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC - 13, pp. 826 - 834, Sept./Oct. 1983.

[16] Grossberg,S., "Classical and instrumental learning by neural networks," *Progress in Theoretical Biology*, vol. 3, pp. 51-141, New York: Academic Press, 1974

[17] Grossberg,S., *Studies of the mind and brain: neural principles of learning perception, development, recognition and motor control*. Dodrecht: D.Reidel, 1982.

[18] Grossberg, S., "Unitization, automaticity, temporal order, and word recognition," *Cognition Brain Theory*, vol. 7, pp. 263-283, 1984.

[19] Grossberg,S. and Kuperstein,M., *Neural dynamics of adaptive sensory motor control: ballistic eye movements*. Amsterdam: North-Holland, 1986.

[20] Grossberg,S. and Stone,G.O., "Neural dynamics of word recognition and recall: attentional priming, learning, and resonance," *Pyschol. Rev.*, vol. 93, pp. 46-74, 1986.

[21] Grossberg,S. and Stone,G.O, "Neural dynamics of attention switching and temporal order information in short term memory," *Memory and Cognition*, 1986.

[22] Hopfield,J.J., "Neural networks and Physical systems with emergent collective computational abilities," *Proc. National Acad. Sci., USA*, vol 79, pp. 2554-2558, April 1982.

[23] Hopfield,J.J., "Neurons with graded response have collective computational abilities," *Proc. National Acad. Sci., USA*, vol 81, pp. 3088-3092, May 1984.

[24] Judd,J.S., *Neural network design and complexity of learning*. Cambridge, MA: MIT Press, 1990.

[25] Koch,C. and Segav,I., *ed., Methods in neuronal modeling: from syanpses to networks*. Cambridge, MA: MIT Press, 1990.

[26] Kohonen,T., *Associative memory: A system-theoretical approach*, New York: Springer-Verlag, 1977.

[27] Kohonen,T., "Adaptive, associative, and self-organisation functions in neural computing," *Applied Optics*, vol. 26, pp. 4910-4918, 1987.

[28] Kosko,B., "Adaptive bi-directional associative memories," *Applied Optics*, vol. 26, No. 23, 1 Dec 1987.

[29] Lippmann,R.P., "An introduction to neural computing with neural nets," *IEEE ASSP Magazine*, vol. 4, April 1987.

[30] Lyon,R.F. and Loeb,E.P., "Isolated digit recognition experiments with a cochlear model," *Proc. ICASSP - 87*, Dallas, Texas, April 1987.

[31] Martin,T. "Acoustic recognition of a limited vocabulary in continuous speech," Ph.D dissertation, Dept. of EE, Univ. Penn., Philadelphia, 1970.

[32] Minsky,M. L. and Papert,S., *Perceptrons*, Cambridge, MA: MIT Press, 1969.

[33] Mueller,P. and Lazzaro,J., "A machine for neural computation of acoustical patterns with application to real-time speech recognition," *AIP Conf. Proc.* 151 *Neural NETS* 1986, Snowbird, Utah, 1986.

[34] Nadel,L., et.al., ed., *Neural computations, mental computation*. Cambridge, MA: MIT Press, 1990.

[35] Narendra,K.S. and Parthasarathy,K., "A diagrammatic representation of back-propagation," Center for Sys. Sci., Dept. Elect. Engr., Yale Univ., New Haven, CT, tech. rep. 8815, Aug. 1988.

[36] Narendra,K.S. and Parthasarathy,K., "Neural networks and dynamical systems. Part I: A gradient approach to Hopfield networks," Center for Sys. Sci., Dept. Elect. Engr., Yale Univ., New Haven, CT, tech. rep. 8820, Oct. 1988.

[37] Narendra,K.S. and Parthasarathy,K., "Neural networks and dynamical systems. Part II: Identification," Center for Sys. Sci., Dept. Elect. Engr., Yale Univ., New Haven, CT, tech. rep. 8902, Feb. 1989.

[38] Narendra,K.S. and Parthasarathy,K., "Back-propagation in dynamical systems containing neural networks," Center for Sys. Sci., Dept. Elect. Engr., Yale Univ., New Haven, CT, tech. rep. 8905, Mar. 1989.

[39] Narendra,K.S. and Parthasarathy,K., "Neural networks and dynamical systems. Part II: Control," Center for Sys. Sci., Dept. Elect. Engr., Yale Univ., New Haven, CT, tech. rep. 8909, Mar. 1989.

[40] Narendra,K.S. and Parthasarathy,K., "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 4-27, Mar. 1990.

[41] Pearlmutter,B., "Learning state-space trajectories in recurrent networks," *Proc. IEEE ICJNN* 89, Washington D.C., 1989, pp. II-365. Publication IEEE TAB neural network committee, 1989.

[42] Rabiner,L.R., "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, no. 2, pp. 257-286, Feb. 1989.

[43] Rabiner,L.R. and Juang,B.H., "An introduction to hidden Markov models," *IEEE ASSP Mag.*, vol 3., no. 1, pp. 4-16, 1986.

[44] Rabiner,L.R. and Sambur,M.R., " An algorithm for determining the endpoints of isolated utterances," *Bell Syst. Tech. J.*, vol. 54, pp. 297-315, Feb 1975.

[45] Rohwer,R., "The *'Moving Targets'* training algorithm," *Neural Networks EURASIP workshop* 1990, Sesimbra, Portugal. Springer-Verlag, 1990, Lecture notes in computer science, vol 412.

[46] Rohwer,R. and Forrest,B., "Training time dependence in neural networks," *Proc. IEEE ICNN*, San Diego, 1987, pp. II - 701.

[47] Rosenblatt,F., *Principles of neuro-dynamics*, Washington, D.C:Spartan, 1962.

[48] Rummelhart,D.E, Hinton,G.E and Williams,R.J., "Learning internal representations by error propagation," in *Parallel distributed processing vol.* 1, Cambridge MA: MIT Press, 1986.

[49] Sejnowski,T. and Rosenberg,C.S., "NETtalk: A parallel network that learns to read aloud," Johns Hopkins Univ. Tech. rep. JHU/EECS - 86/01, 1986.

[50] Tank,D.W. and Hopfield,J.J., "Computing with neural circuits: a model," *Science*, vol 233, pp. 625-633, Aug. 1986.

[51] Tank,D.W. and Hopfield,J.J., "Neural computation by concentrating information in time," *Proc. National Acad. Sci., USA*, pp. 1896-1900, April 1987.

[52] Waibel,A., Hanazawa,T., Hinton,G., Shikamo,K. and Lang,K.J., "Phoneme recognition using time-delay neural networks," *IEEE ASSP*, vol. 37, No. 3, pp. 328 - 339, March 1989.

[53] Waibel,A. and Yegnanarayana,B., "Comparative study of non-linear time warping techniques in isolated word speech recognition systems," Tech. report, Carniege-Mellon Univ., June 1981.

[54] Wasserman,P.D., *Neural computing: theory and practice*. New York: Van Nostrand Reinhold, 1989.

[55] Werbos,P., *Energy models and studies*, B. Lev, Ed., North Holland, 1983.

[56] Zornetzer,S.F., Davis,J.L. and Lau C., An introduction to neural and electronic networks. San Deigo: Academic Press, 1990.